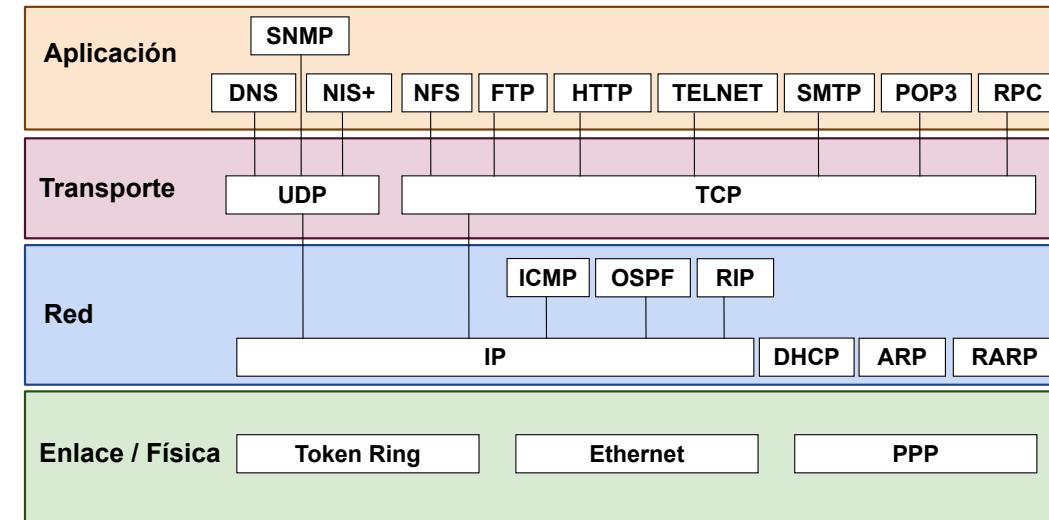




## TEMA 1.1. Revisión de IPv4. Protocolo DHCP

### PROFESORES:

Rubén Santiago Montero  
 Eduardo Huedo Cuesta  
 Rafael Rodríguez Sánchez  
 Luis M. Costero Valero



## Protocolo de Internet: IP

### Protocolo de red de TCP/IP

- Proporciona un servicio básico de entrega de paquetes
- Protocolo **no orientado a conexión** (no fiable)
  - No realiza detección ni recuperación de paquetes perdidos o erróneos
  - No garantiza que los paquetes lleguen en orden
  - No garantiza la detección de paquetes duplicados

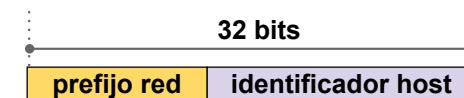
### Funciones básicas del protocolo IP

- Esquema global de direccionamiento
  - Dirección IP
- Encapsulado de datos y formato
  - Datagrama IP
- Fragmentación y reensamblaje de paquetes
  - División del paquete en fragmentos de un tamaño aceptable por la red
- Reenvío de paquetes
  - Retransmisión de paquetes de una red a otra, basada en la información de la tabla de rutas, que se construye con los protocolos de encaminamiento (RIP, OSPF, BGP...)

## Direccionamiento IP

### Estructura y Notación

- Las direcciones IP constan de 4 bytes (32 bits)
- Para expresarlas se utiliza la “notación de punto” (Ej. 10.0.0.1)



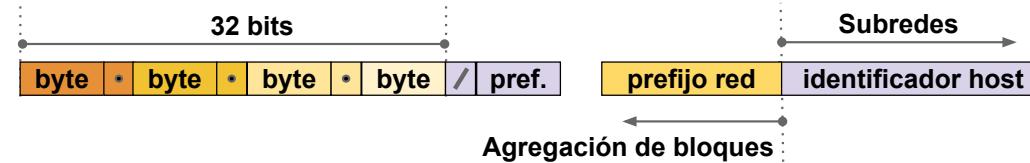
### Direccionamiento basado en clases

	8 bits	24 bits	
Clase A	0 red	host	[0-127].x.x.x
Clase B	10 red	host	[128-191].[0-255].x.x
Clase C	110 red	host	[192-223].[0-255].[0-255].x
Clase D	1110	grupo multicast	[224-239].x.x.x

# Direccionamiento IP

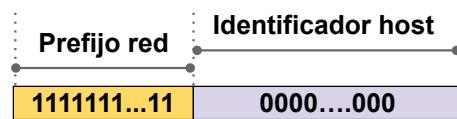
## CIDR (Classless Inter-Domain Routing)

- Intenta aliviar el problema del agotamiento de direcciones
- Elimina la estructura fija basada en clases
- El espacio de direcciones se divide en bloques de tamaño arbitrarios
- Notación barra (o CIDR) que incluye la longitud del prefijo



## Máscara de Red

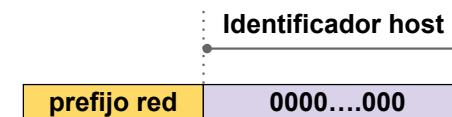
- Sirve para determinar el prefijo de red de una dirección IP (Y lógica)
- Notación decimal o CIDR



# Direccionamiento IP

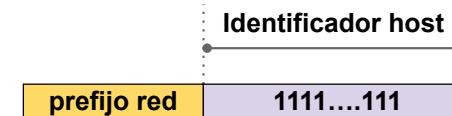
## Direcciones de Red

- Se utilizan para representar a una red completa en las tablas de rutas
- Nunca se utilizan como dirección destino ni se asignan a un host concreto



## Direcciones de Broadcast

- Se utilizan para enviar un paquete a todas las máquinas de la red local



## Direcciones de Loopback

- Direcciones de bucle interno
- Red 127.0.0.0/8 (típicamente 127.0.0.1)

# Direccionamiento IP

## Ejemplo

Un computador tiene la siguiente dirección IP en notación CIDR: 150.26.193.66/21.

Determinar:

1. La máscara de red en notación decimal de punto.
2. La dirección de la red.
3. La dirección de difusión (broadcast) de la red.
4. El número de direcciones útiles para las máquinas de la red.
5. El rango de direcciones IP que pueden asignarse a las máquinas de la red.

Calculadora IP, subnetting/supernetting: <http://jodies.de/ipcalc>

# Direccionamiento IP

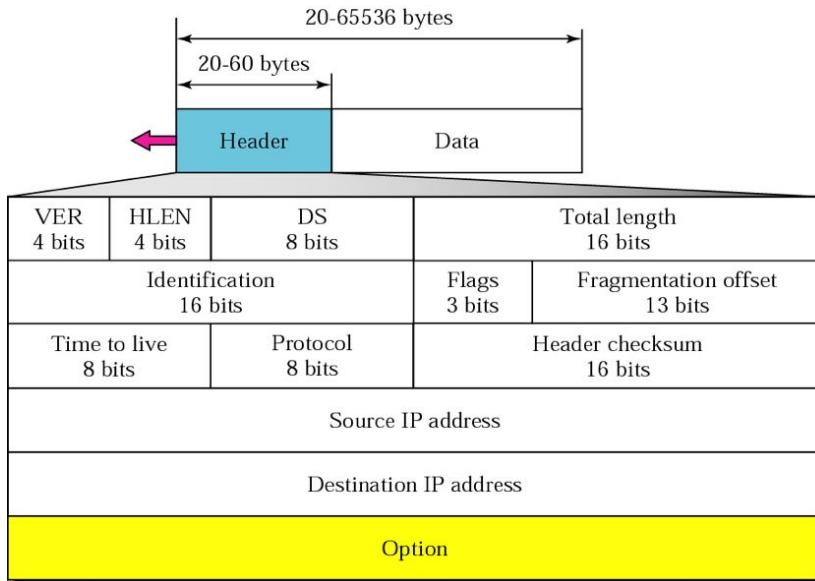
## Direcciones Privadas

- Conjunto de direcciones reservadas para uso privado
- No son válidas para su uso en Internet
- Los rangos de direcciones IP privadas son los siguientes:
  - 10.0.0.0 – 10.255.255.255 ~ 1 red privada de clase A (/8)
  - 172.16.0.0 – 172.31.255.255 ~ 16 redes privadas de clase B (/16)
  - 192.168.0.0 – 192.168.255.255 ~ 256 redes privadas de clase C (/24)

## Direcciones Multicast (224.0.0.0/4) - RFC 1112

- Identifican de forma lógica a un grupo de hosts en el segmento de red. Ejemplos
  - 224.0.0.1 (todos los hosts)
  - 224.0.0.2 (todos los routers)
  - 224.0.0.251 (mDNS)
- Relación con la capa de enlace (Ethernet - tipo 0x0800, RFC 7042 - Sección 2.1.1)
  - IP: 224.0.0.1 → 23 bits
  - MAC: 01:00:5E:00:00:01

# Formato del datagrama IP



# Traducción de direcciones: ARP

- ARP (Address Resolution Protocol) correspondencia entre direcciones IP y MAC
- 
- Datagrama IP:
- |            |           |       |  |
|------------|-----------|-------|--|
| IP Destino | IP Fuente | Datos |  |
|------------|-----------|-------|--|
- Trama Ethernet:
- |             |            |      |       |
|-------------|------------|------|-------|
| MAC Destino | MAC Fuente | Tipo | Datos |
|-------------|------------|------|-------|

- La tabla ARP mantiene las direcciones IP de las últimas máquinas con las que nos hemos comunicado y las direcciones Ethernet asociadas

## Formato del mensaje ARP

0	8	16	31
Hardware Type			Protocol Type
Hardware length	Protocol length	Operation <b>Request:1, Reply:2</b>	
Source hardware address			
Source protocol address			
Destination hardware address (Empty in request)			
Destination protocol address			

# Reenvío de Paquetes

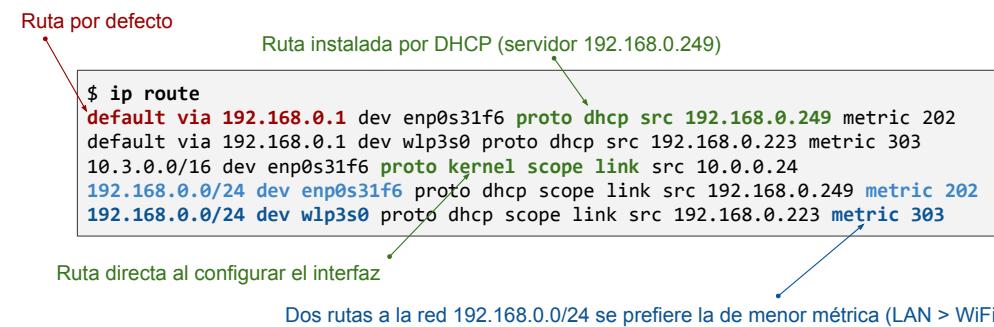
- Cuando un host tiene que enviar un paquete, lo hace por el enlace adecuado para alcanzar el destino usando la tabla de rutas
- Reenvío basado en dirección destino.** Se busca un destino que coincida y se reenvía por esa ruta (no orientado a conexión)
  - Entradas en la tabla (rutas) por host, red o por defecto
  - Las entradas de red pueden ser con o sin clase
- Reenvío basado en etiqueta.** Cada paquete se etiqueta y se reenvía según esa etiqueta (orientado a conexión)
  - Reduce la complejidad de la tabla de rutas
  - Se usa siempre el mismo circuito (entrega en orden y tiempo predecible)
  - Campo Flow Label en la cabecera IPv6 y MPLS (Multiprotocol Label Switching)

# Reenvío Basado en Dirección Destino

- La tabla de rutas tiene información sobre:
  - Destino
  - Máscara de red o longitud de prefijo (para direccionamiento sin clase)
  - Interfaz (para entrega directa) y/o siguiente salto (para entrega indirecta)
  - Métrica (preferencia de ruta)
- El destino puede ser
  - Un host específico
  - Una red
  - Default* ( $0.0.0.0/0$ ), para paquetes que no encajen en ninguna entrada
- Proceso de selección de destino:
  - Buscar la ruta más específica que encaje con la dirección destino (*longest match prefix*)
  - Si hay más de una ruta con igual especificidad, elegir la de menor métrica

# Reenvío Basado en Dirección Destino

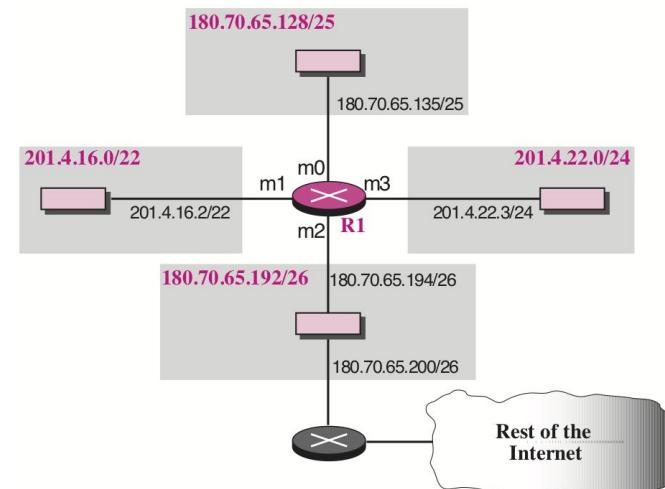
- La tabla de rutas de un host contiene:
  - Ruta por defecto**, que se establece en el proceso de autoconfiguración (ver DHCP) o de forma manual (`ip route add default`)
  - Rutas directas** a las redes configuradas en cada interfaz
  - Rutas específicas** configuradas de forma manual (`ip route add <dest>`)



# Reenvío: Ejemplo

Dada la siguiente topología de red:

- Determine la tabla de rutas para el encaminador R1
- Describir el procesamiento de dos paquetes con dirección destino 201.4.22.35 y 18.24.32.78, respectivamente



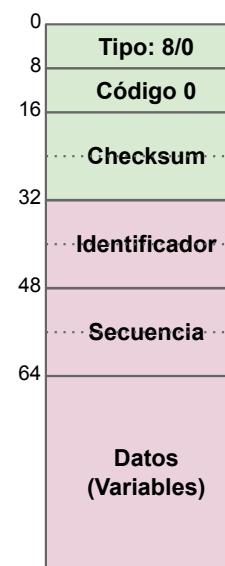
# Protocolo de mensajes de control: ICMP

## ICMP: Echo Request/Reply

### Características

- Es un protocolo para el intercambio de mensajes de control en la red
- Los mensajes ICMP se pueden clasificar en dos tipos:
  - Error**: para informar de situaciones de error en la red
  - Informativos**: sobre la presencia o el estado de un determinado sistema

Tipo ICMP	Mensajes Error		Mensajes Informativos	
Código	Tipo	Significado	Tipo	Significado
Checksum	3	Destination Unreachable	0	Echo Reply
	4	Source Quench	5	Redirect
	11	Time Exceeded	8	Echo Request
ICMP Datos (Opcionales)	12	Parameter Problem	9	Router Solicitation
			10	Router Advertisement

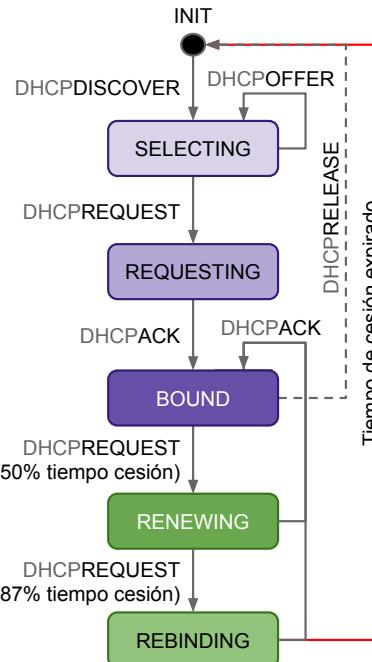


- Se utilizan para ver si un computador es alcanzable
- Formato de los mensajes Echo Request/Echo Reply
  - Identificador**: Permite establecer la correspondencia entre solicitud (Request) y respuesta (Reply); ambos con el mismo identificador.
  - Secuencia**: También se utiliza para establecer la correspondencia entre solicitud y respuesta, cuando se envían varios Echo Requests consecutivos con el mismo identificador.
  - Datos**: Un número determinado de bytes aleatorios.
- La orden ping envía mensajes ICMP Echo Request y espera la recepción de mensajes ICMP Echo Reply

# Configuración dinámica: DHCP

- DHCP (Dynamic Host Configuration Protocol) proporciona configuración automática de los parámetros de la red
    - Dirección IP y máscara de red
    - Router predeterminado
    - Servidores DNS
    - Otros parámetros y servicios de red
  - **Antecedentes**
    - RARP (Reverse ARP): Sólo es útil en el segmento de red. Únicamente provee la dirección IP
    - BOOTP (Bootstrap Protocol): Soluciona los problemas de RARP pero sólo soporta configuraciones estáticas (similar a DHCP en configuración estática)
  - **Características (RFC 2131)**
    - Protocolo cliente/servidor sobre UDP en los puertos 67 (servidor) y 68 (cliente). **Nota:** el puerto cliente no es un puerto efímero
    - Control de errores basado en sumas de comprobación, temporizadores y retransmisiones
    - Protocolo TFTP para la transferencia de ficheros con información adicional o imágenes de arranque
    - DHCP Relay Agent para servidores/clientes en diferentes redes

# DHCP: Diagrama de estados y mensajes



## DHCP: Formato del mensaje

0	8	16	24	3
Operation code	Hardware type	Hardware length	Hop count	
Number of seconds		Flags		
		Client IP address		
		Your IP address		
		Server IP address		
		Gateway IP address		
Client hardware address (16 bytes)				
		Server name (64 bytes)		
		Boot file name (128 bytes)		
		Options (Variable length)		
Tag	Length	Value (Variable length)		

**Code:** 0x01 (request), 0x02 (reply)

**Hw type - length:** 1 - 6 para Ethernet

**Trans. ID:** Correspondencia entre solicitud y respuesta

Your IP: ofrecida por el servidor

**Server name - Boot filename:**  
compatibilidad con BOOTP

**Options:** Información de configuración (RFC 2132)

- Servidores DNS
  - Host name
  - TCP/IP (MTU, TTL...)
  - Servidores NTP, SMTP, POP3...
  - DHCP extensions (tipo mensaje, servidor TFTP, tiempo de cesión, Id. servidor, Id. cliente...)

# Ejemplos de Preguntas Teóricas

¿Cuál de los siguientes parámetros NO se puede configurar usando DHCP?

- Dirección física (MAC)
  - Dirección de red (IP)
  - Router predeterminado

¿Qué mensaje DHCP se usa para realizar una petición de oferta por parte del cliente?

- DHCPDISCOVER
  - DHCPOFFER
  - DHCPREQUEST

Con la introducción de CIDR, se pretende...

- ...ampliar el espacio total de direcciones.
  - ...dividir el espacio de direcciones en bloques de tamaño fijo.
  - ...aliviar el problema del agotamiento de direcciones.



## El protocolo TCP: Características

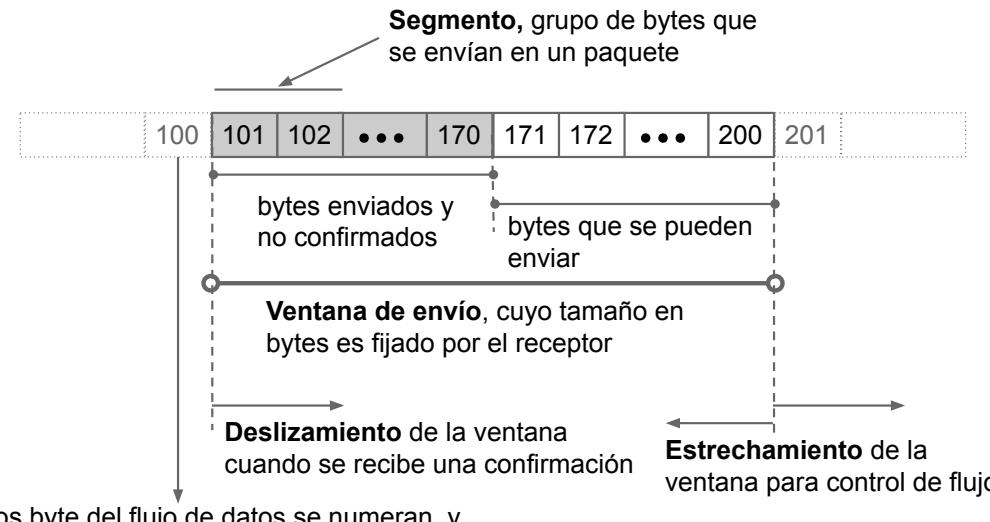
### TEMA 1.2. Conceptos Avanzados del Protocolo TCP

#### PROFESORES:

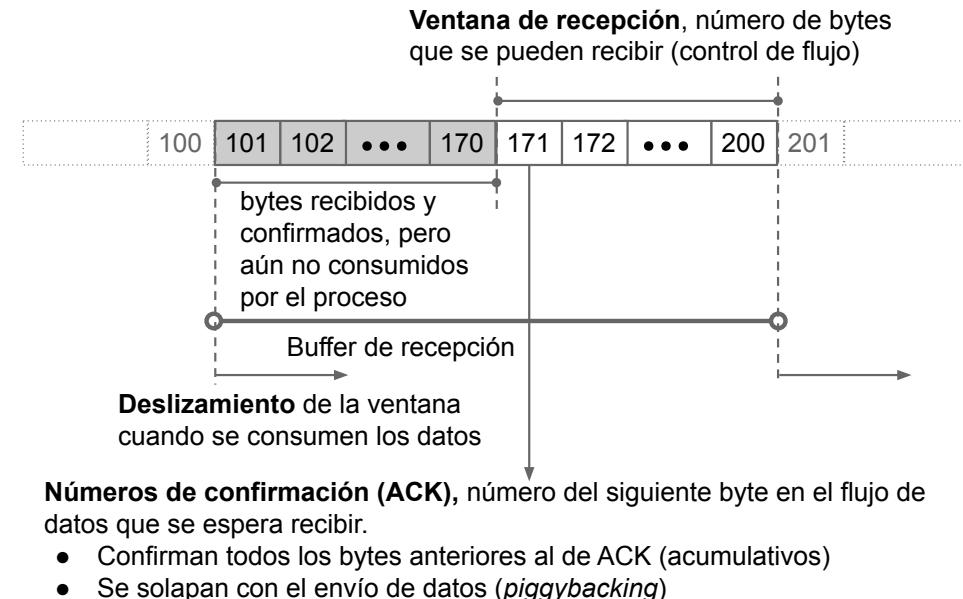
Rubén Santiago Montero  
Eduardo Huedo Cuesta  
Rafael Rodríguez Sánchez  
Luis M. Costero

- Servicios ofrecidos por TCP:
  - Comunicación lógica proceso-proceso, usando números de puerto
  - Transferencia como flujo de bytes (*byte stream*)
  - Transmisión orientada a conexión y fiable
  - Full-duplex y multiplexación
- Orientado a conexión, define las siguientes fases para la transmisión:
  - Establecimiento de conexión
  - Transferencia de datos
  - Cierre de conexión
- Unidad de transferencia: Segmento TCP
- Fiable, incluye mecanismos de control de errores de tipo ventana deslizante con:
  - Códigos de comprobación (*checksum*)
  - Numeración de segmentos
  - Confirmaciones selectivas y acumuladas, superpuestas del receptor
  - Retransmisión de segmentos perdidos o erróneos
  - Temporizadores
- [Especificado por RFC 9293](#)

### Ventana Deslizante: La Ventana de Envío



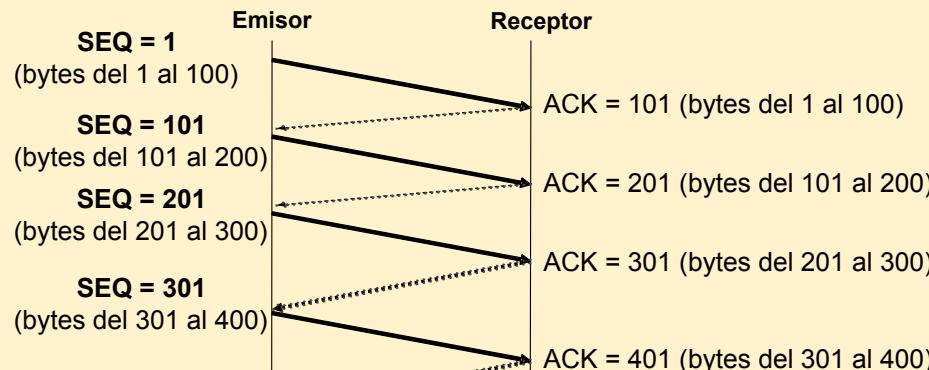
### Ventana Deslizante: La Ventana de Recepción



# Ventana Deslizante: Funcionamiento

**Ejemplo:** Transmisión sin errores.

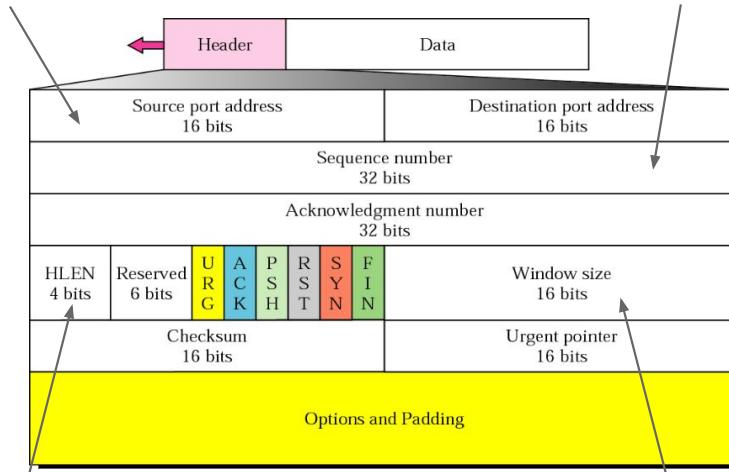
Tamaño de la ventana = 100 bytes, Tamaño del segmento = 100 bytes.



**Ejemplo:** ¿Cuáles serían los números de secuencia y confirmación para un tamaño de segmento de 50 bytes? (Nota, las confirmaciones en TCP no tienen que realizarse de forma inmediata)

# Formato del Segmento TCP

Puertos, identifican los extremos de la conexión



Longitud de la cabecera en palabras de 32 bits (20-60 bytes)

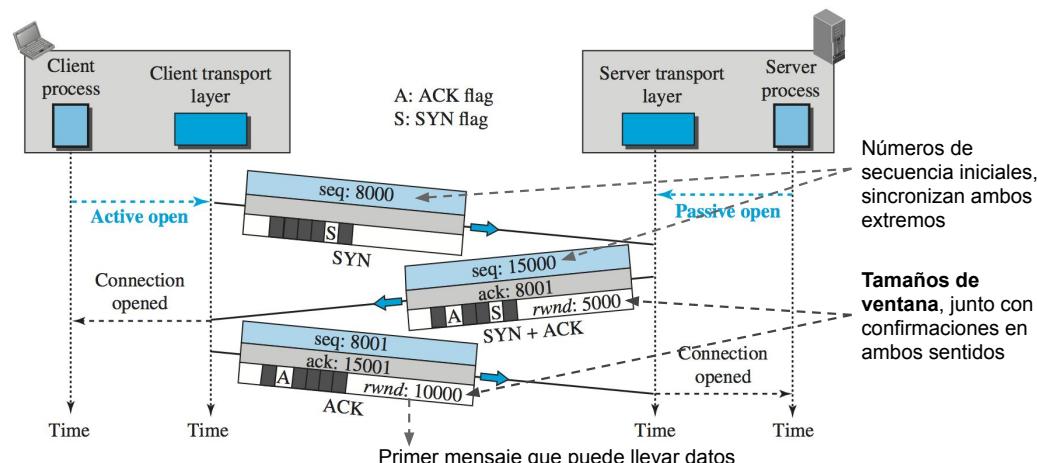
**Tamaño de la ventana** en bytes (control de flujo)

# Formato del Segmento TCP

## Flags del campo de control (6 bits)

- SYN:** Utilizado en el establecimiento de la conexión y sincronizar los números de secuencia iniciales
- FIN:** Utilizado en la finalización de la conexión
- ACK:** El segmento contiene un número de confirmación válido (ACK=1). Todos los segmentos de una conexión TCP, excepto el primero, llevan ACK=1
- RST:** Utilizado para denegar o abortar una conexión
- PSH:** Los datos deben ser entregados inmediatamente a la aplicación (PSH=1), o pueden almacenarse en el buffer (PSH=0)
- URG:** El segmento transporta datos urgentes (URG=1) desde el primer byte hasta el nº de byte especificado en el campo **Urgent pointer**
  - TCP notifica a la aplicación de los datos urgentes (SIGURG)
  - El tratamiento de *urgencia* corresponde a la aplicación, no a TCP

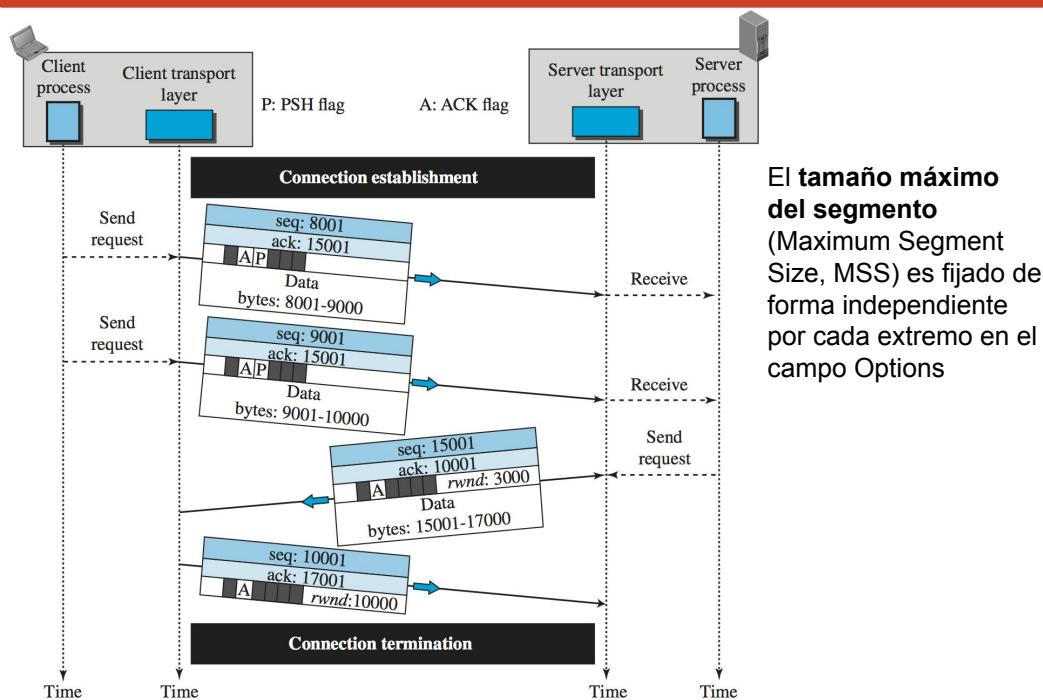
# Fases de la Conexión: Establecimiento (3-way)



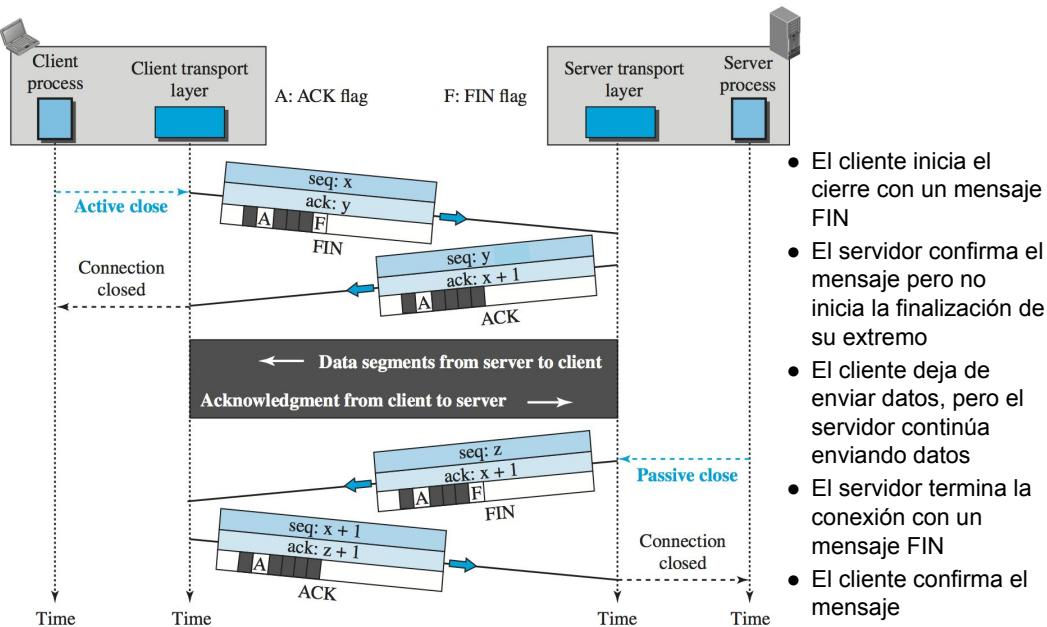
## Ataque TCP SYN Flooding

- Se envía una gran cantidad de segmentos TCP con el flag SYN activado, que consumen recursos del servidor y puede llegar a no responder (DoS)
- Contramedidas: filtrar conexiones (limitar tasa o detectar IPs suplantadas), aumentar recursos o retrasar la asignación de recursos (usando SYN cookies)

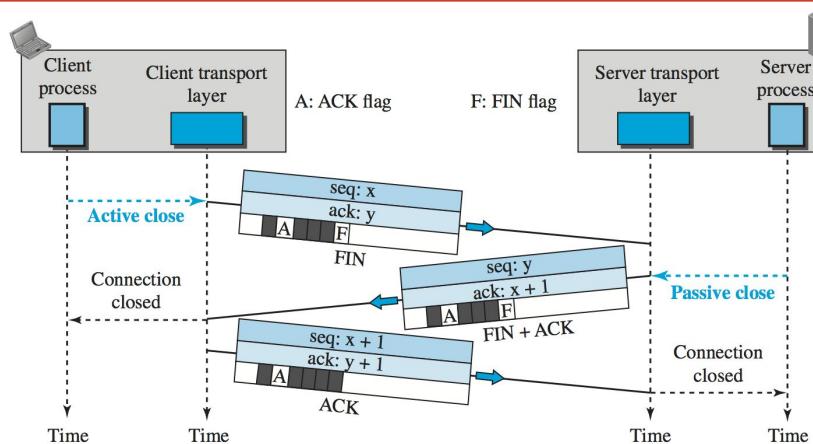
## Fases de la Conexión: Transferencia



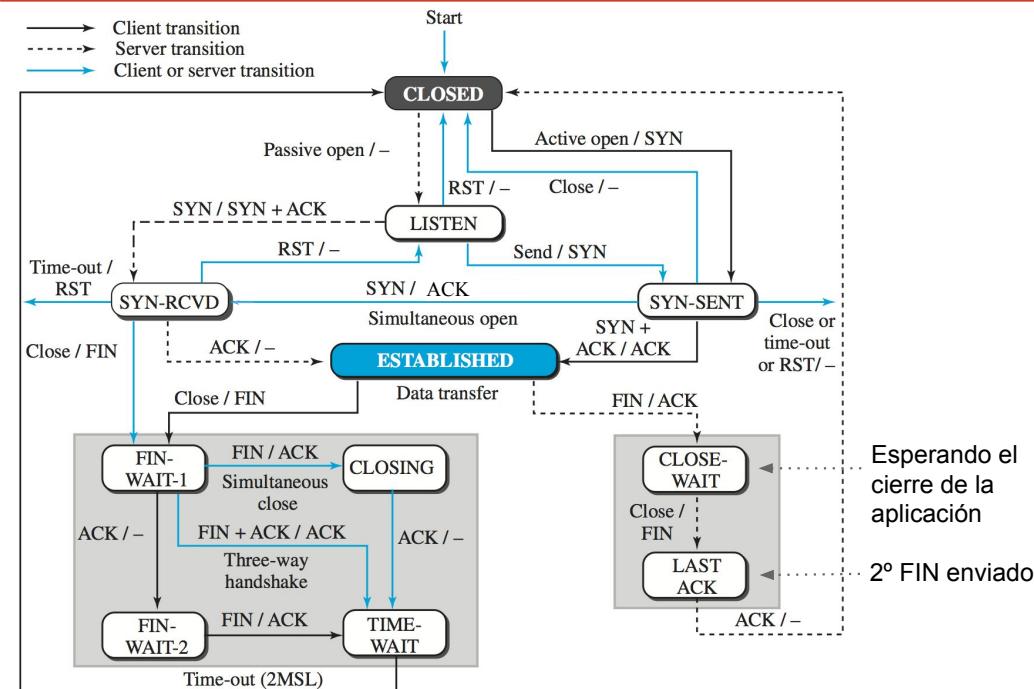
## Fases de la Conexión: Finalización (4-way)



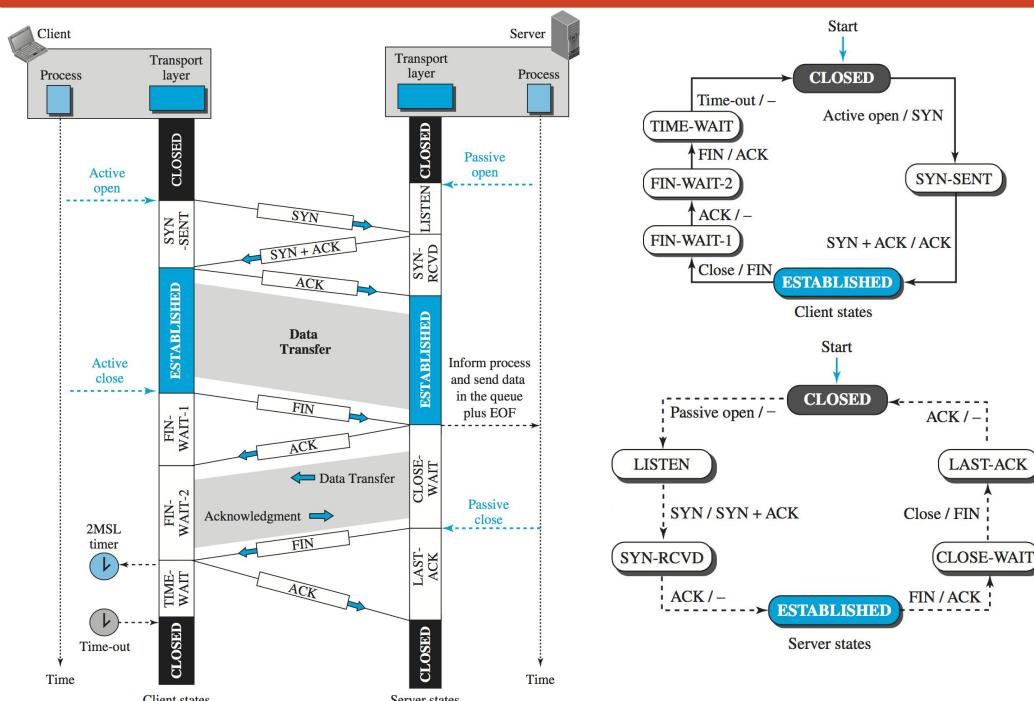
## Fases de la Conexión: Finalización (3-way)



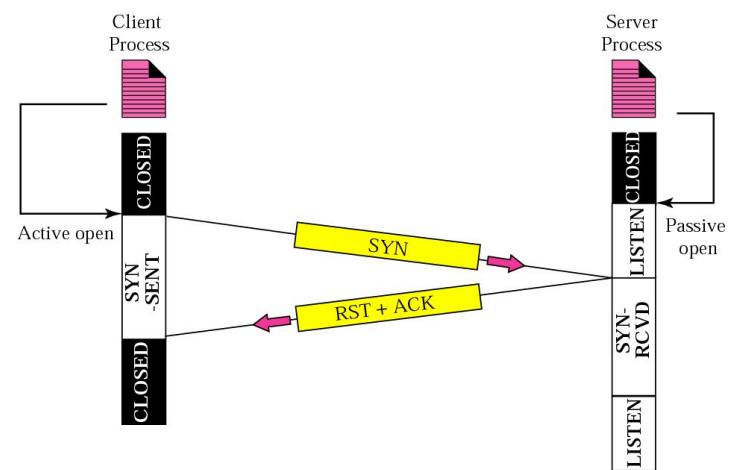
## Fases de la Conexión: Máquina de Estados



## Fases de la Conexión: Máquina de Estados



## Fases de la Conexión: Máquina de Estados



**Ejemplos:** Describir la secuencia de estados para el cliente y servidor durante:

- El cierre de tres vías
- El cierre simultáneo

## Control de Errores: Confirmaciones

- El control de errores se realiza usando el mecanismo de ventana deslizante que permite gestionar:
  - La retransmisión de segmentos erróneos o perdidos
  - La recepción de segmentos duplicados o fuera de orden

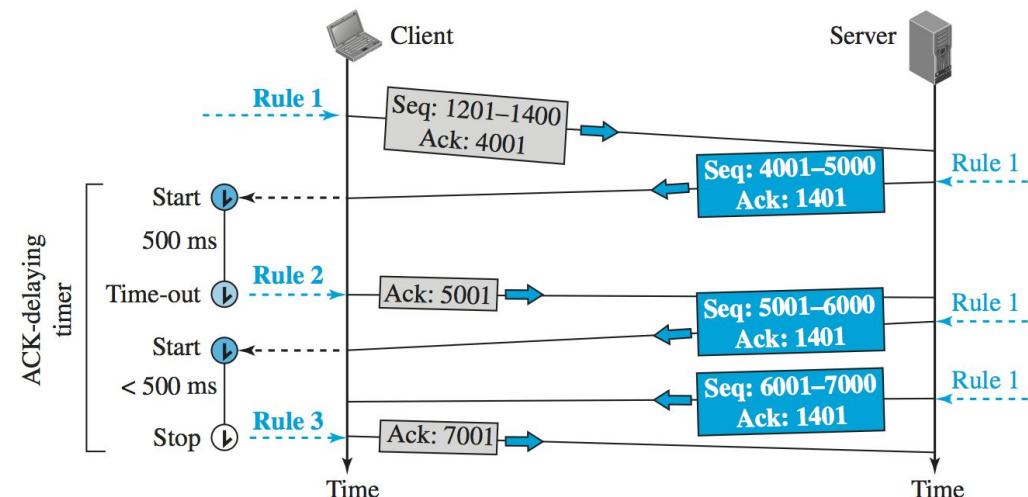
### Reglas para las confirmaciones

1. Los segmentos de datos deben incluir una confirmación (*piggyback*) indicando el siguiente número de secuencia esperado, que confirma todos los anteriores
2. Si no hay datos que enviar, las confirmaciones de segmentos recibidos en orden pueden retrasarse 500 ms para incluirlas en un segmento de datos
3. Solo puede retrasarse la confirmación de un segmento
4. Los segmentos recibidos fuera de orden se confirman inmediatamente (esto provocará una retransmisión rápida, como veremos)
5. Los segmentos que completan huecos se confirman inmediatamente
6. Los segmentos duplicados se confirman para solucionar la pérdida de un ACK

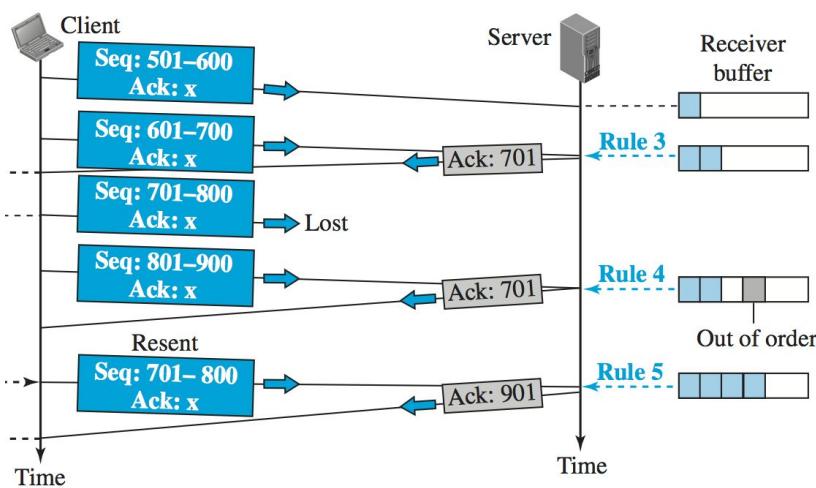
### Opcional

- Confirmaciones selectivas (SACK) de segmentos fuera de orden
  - No reemplazan los ACK, informativos para el emisor
  - Implementados como opción TCP

## Control de Errores: Transmisión sin Error



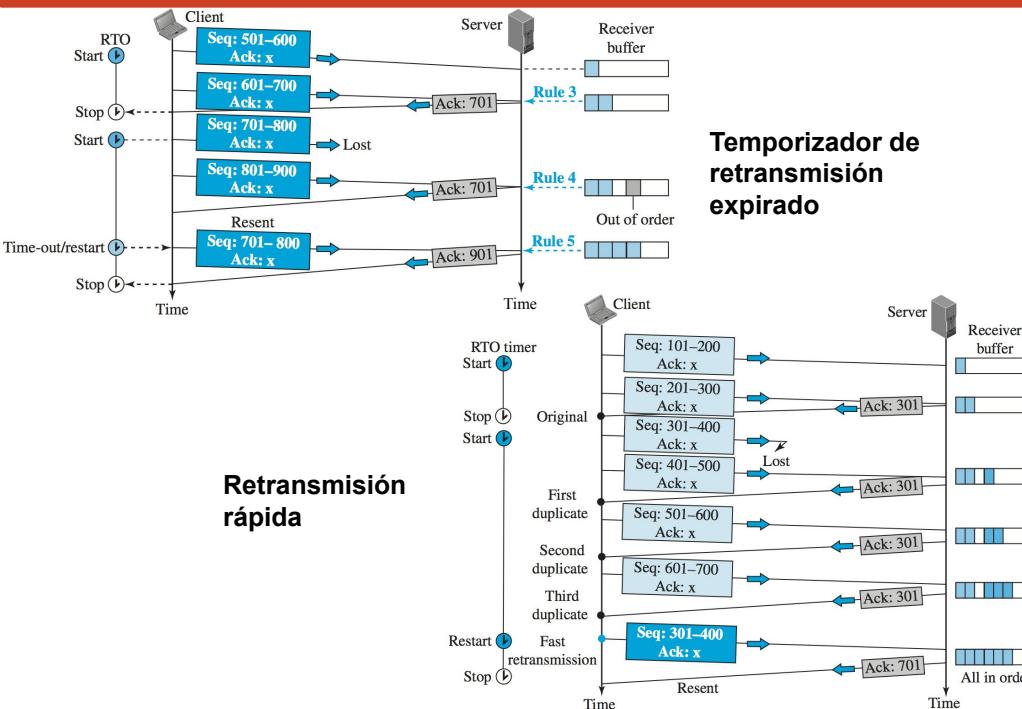
## Control de Errores: Recepción fuera de orden



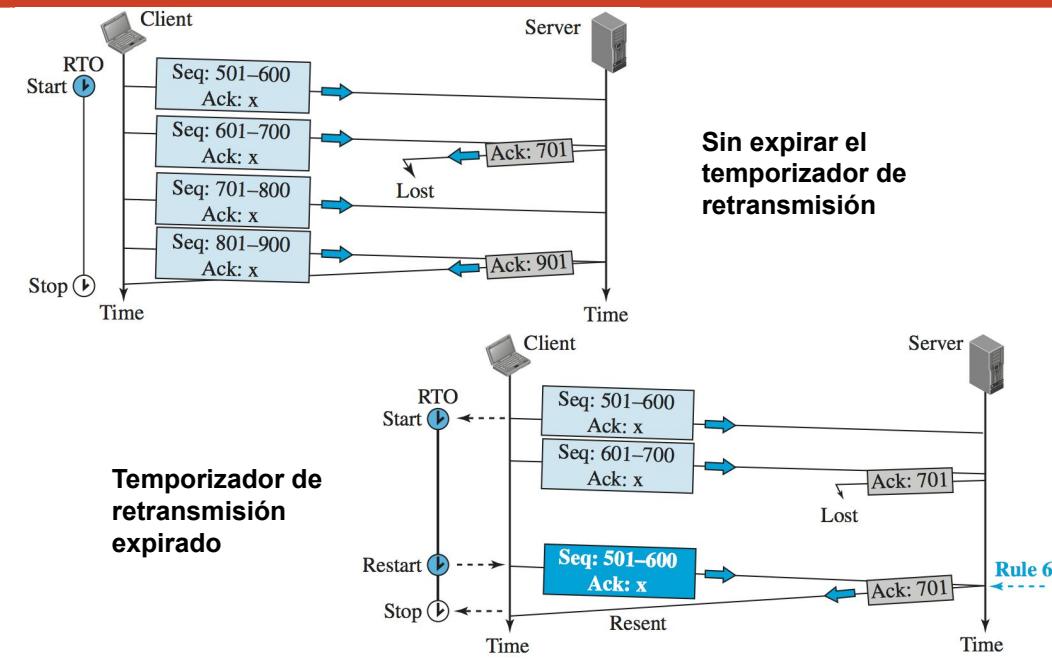
## Control de Errores: Retransmisión

- La capacidad para volver a transmitir un segmento TCP cuando no se recibe o se recibe erróneamente es el núcleo del control de errores
- TCP dispone de dos mecanismos de retransmisión:
  - Temporizador de retransmisión**
    - Se inicia cuando se envía un segmento y se para al recibir la confirmación
    - Si expira, se retransmite el primer segmento sin confirmar de la ventana
    - Existen diversos algoritmos para fijar el RTO (Retransmission Time-Out), que debe ser mayor que el RTT (Round-Trip Time)
    - Cada conexión debe usar un único temporizador (ver RFC 6298)
  - Retransmisión rápida**
    - Se retransmite cuando se reciben 3 ACKs duplicados
    - No requiere que expire el temporizador de retransmisión

## Control de Errores: Pérdida de un Segmento



## Control de Errores: Pérdida de un ACK



# Temporizadores TCP

- **Retransmisión**
- **Keepalive**, que evita mantener conexiones indefinidamente
  - Una conexión puede estar en silencio `tcp_keepalive_time` segundos
  - Despues, se envía un máximo de `tcp_keepalive_probes` sondas cada `tcp_keepalive_intvl` segundos
  - Si no se recibe ningún ACK para las sondas, se cierra la conexión
  - Ej. 2 horas, 10 sondas cada 75 segundos
- **TIME-WAIT**, útil en dos situaciones:
  - Reenviar el último ACK durante el cierre activo si se retransmite FIN
  - Permitir que expiren los posibles segmentos duplicados
  - Impide una nueva conexión con los mismos parámetros (direcciones y puertos origen y destino)
  - $2 \times MSL$  (Maximum Segment Lifetime). Ej. 60 segundos
- **Temporizador de persistencia**, asociado al anuncio de un tamaño de ventana 0
  - Recupera la pérdida de un ACK posterior con el nuevo tamaño
  - Se envía una sonda que fuerza el envío de un ACK
  - Ej. 60 segundos

# Temporizador de Retransmisión

- La elección del tiempo de vencimiento del temporizador de retransmisión (RTO) está basada en los retardos observados en la red (RTT)
- Los retardos en la red pueden variar dinámicamente, por tanto el temporizador debe adaptarse a esta situación
- Las principales técnicas utilizadas para fijar el temporizador de retransmisión son las siguientes:
  - Método de la media ponderada (algoritmo de Jacobson)
  - Método de la varianza (algoritmo de Jacobson/Karels)
  - Algoritmo de Karn

## Temporizador de Retransmisión

### Tiempo de ida y vuelta medido ( $RTT_M$ )

- Cuando se envía segmento, se mide el tiempo transcurrido desde que se envía el segmento hasta que se recibe el ACK, denominado  $RTT_M$  (Measured Round-Trip Time)
- Sólo hay un temporizador  $RTT_M$
- El valor del  $RTT_M$  puede experimentar grandes fluctuaciones

### Tiempo de ida y vuelta suavizado ( $RTT_S$ )

- Evitar las fluctuaciones del RTT
- $RTT_S$  (Smoothed Round-Trip Time) es la media ponderada entre el  $RTT_M$  y el último  $RTT_S$  calculado:

$$\text{1ª medida: } RTT_S = RTT_M$$

$$\text{Siguientes: } RTT_S = (1 - \alpha) \times RTT_S + \alpha \times RTT_M, \text{ con } \alpha < 1 \text{ (ej. } \alpha = \frac{1}{2} \text{)}$$

### Desviación del RTT ( $RTT_D$ )

- Para considerar la variación del tiempo de ida y vuelta

$$\text{1ª medida: } RTT_D = RTT_M / 2$$

$$\text{Siguientes: } RTT_D = (1 - \beta) \times RTT_D + \beta \times |RTT_S - RTT_M|, \text{ con } \beta < 1 \text{ (ej. } \beta = \frac{1}{4} \text{)}$$

## Temporizador de Retransmisión

### Algoritmo de Jacobson

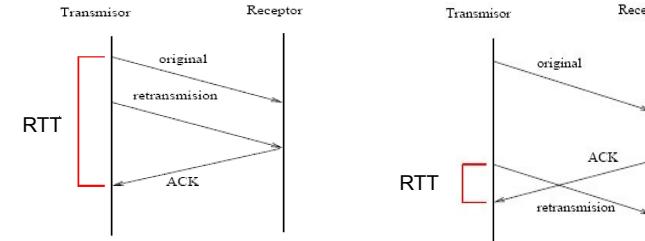
- Considera únicamente el  $RTT_S$
- El RTO se calcula después de cada medida del RTT como  $RTO = \gamma \times RTT_S$  (ej.  $\gamma = 2$ , el doble del RTT estimado)

### Algoritmo de Jacobson/Karels

- Combina  $RTT_S$  y  $RTT_D$   
$$RTO = RTT_S + 4 \times RTT_D$$

### Algoritmo de Karn

- Previene la ambigüedad en el cálculo del  $RTT_M$  cuando hay retransmisiones
- En esos casos, no actualiza  $RTT_S$  y  $RTT_D$  y duplica el RTO (**exponential backoff**)

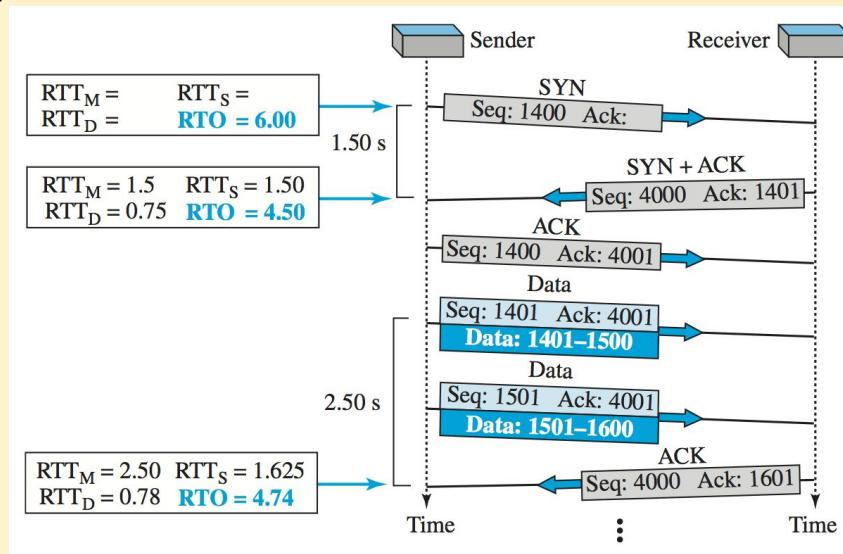


# Temporizador de Retransmisión

# Temporizador de Retransmisión

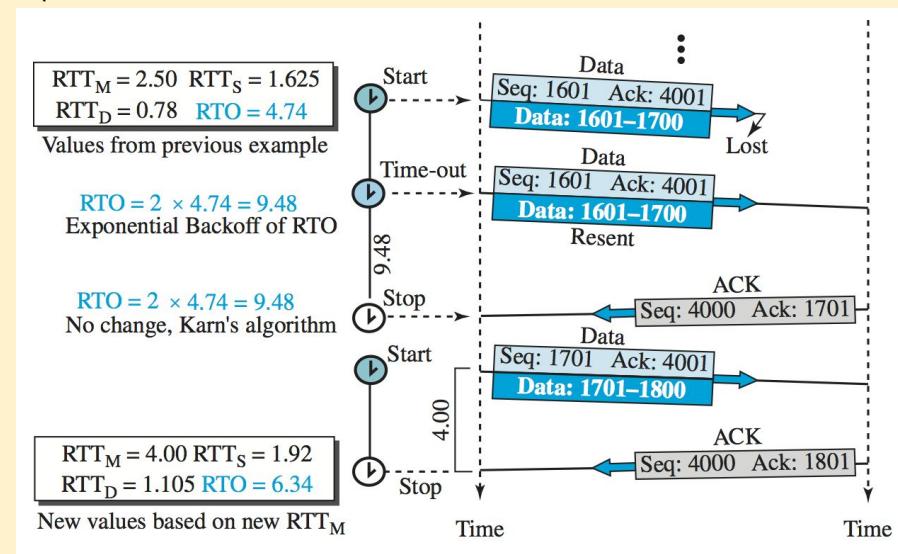
**Ejemplo:** Calcular los valores de los temporizadores:

- $\alpha=1/8$ ,  $\beta=1/4$



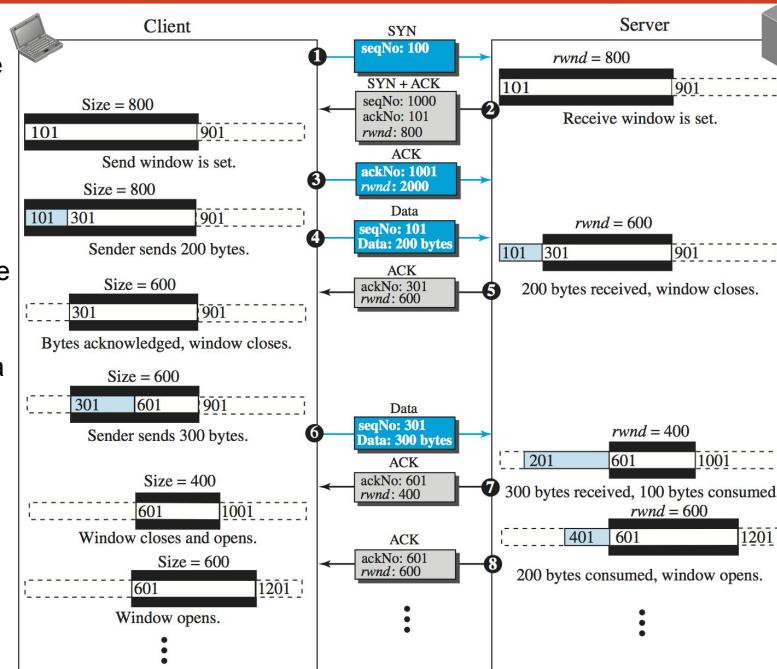
**Ejemplo:** Calcular los valores de los temporizadores:

- $\alpha=1/8$ ,  $\beta=1/4$



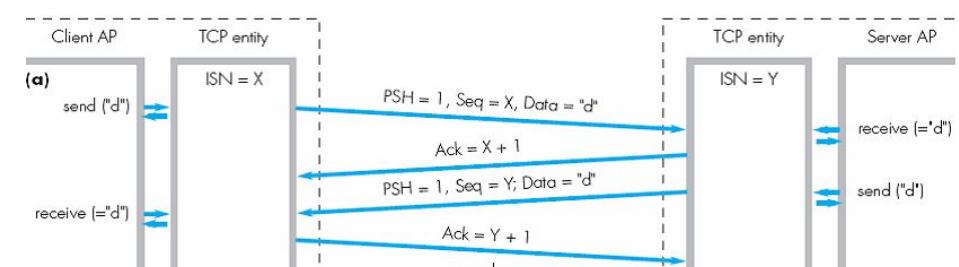
## Control de Flujo

- Controla la tasa de envío de datos para evitar la sobrecarga del receptor
- El control de flujo se realiza mediante la ventana de recepción, anunciada en cada ACK



## Control de Flujo: Síndrome de la ventana trivial

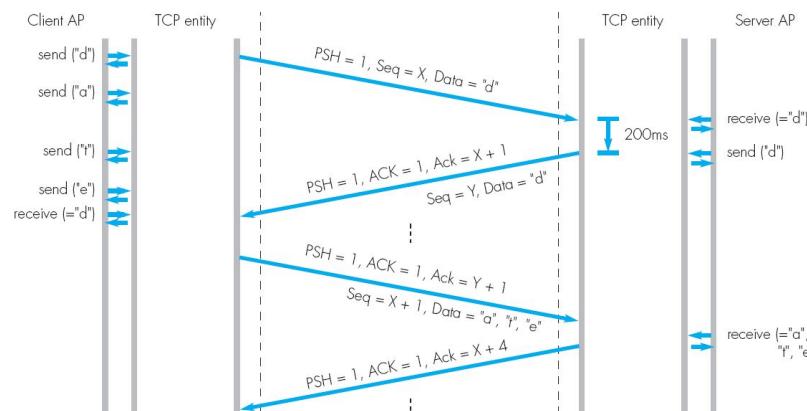
- El síndrome de la ventana trivial (*silly window syndrome*) se produce cuando:
  - La aplicación emisora genera datos a un ritmo muy lento (ej. byte a byte)
  - La aplicación receptora consume datos a un ritmo muy lento
- **Ventana trivial en el emisor** (ej. aplicaciones interactivas)
  - Cada carácter necesita 4 mensajes TCP/IP (40 bytes de cabeceras)
  - Un carácter (1 bytes) usa más de 160 bytes



## Control de Flujo: Síndrome de la ventana trivial

- **Algoritmo de Nagle (RFC 1122, Sec. 4.2.3.4)**

- El emisor envía el primer mensaje (aunque sea un solo byte)
- Los siguientes mensajes se retrasan hasta que:
  - se recibe un ACK del receptor
  - se acumula un segmento completo (MSS bytes) de la aplicación
  - expira el RTO



## Control de Flujo: Síndrome de la ventana trivial

- **Ventana trivial en el receptor**

- La aplicación consume los datos a un ritmo lento
- Se anuncian ventanas de tamaño reducido, produciendo el efecto anterior

- **Algoritmo de Clark**

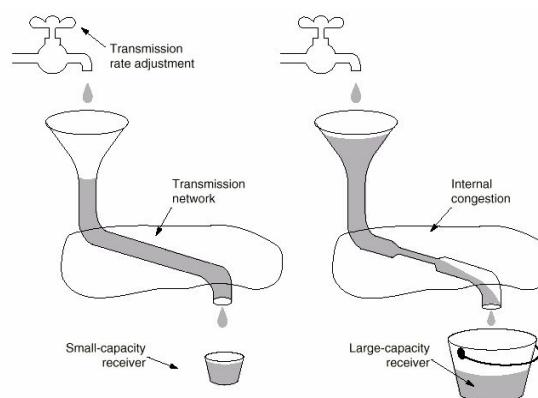
- Anunciar un tamaño de ventana 0 hasta que:
  - Haya espacio para recibir un segmento completo (MSS)
  - Se haya liberado la mitad del buffer de recepción

- **Retrasar los ACKs**

- Para evitar que el emisor desplace la ventana
- Reduce el tráfico (número de ACKs), pero puede provocar retransmisiones innecesarias de segmentos no confirmados
- TCP establece que no deben retrasarse más de 500 ms

## Control de la Congestión

- Cuando se pierden paquetes en Internet, la mayoría de las veces se debe a un problema de congestión en algún punto de la red:
  - El router no puede procesar y reexpedir paquetes al ritmo al que los recibe
  - Cuando el router se satura, empieza a descartar paquetes (incluidas las confirmaciones)
- El control de la congestión y el de flujo son dos mecanismos diferentes:



## Control de la Congestión

- El emisor utiliza el ritmo de llegada de confirmaciones para regular el ritmo de envío de segmentos de datos
- Esto se implementa mediante la **ventana de congestión (CW)**

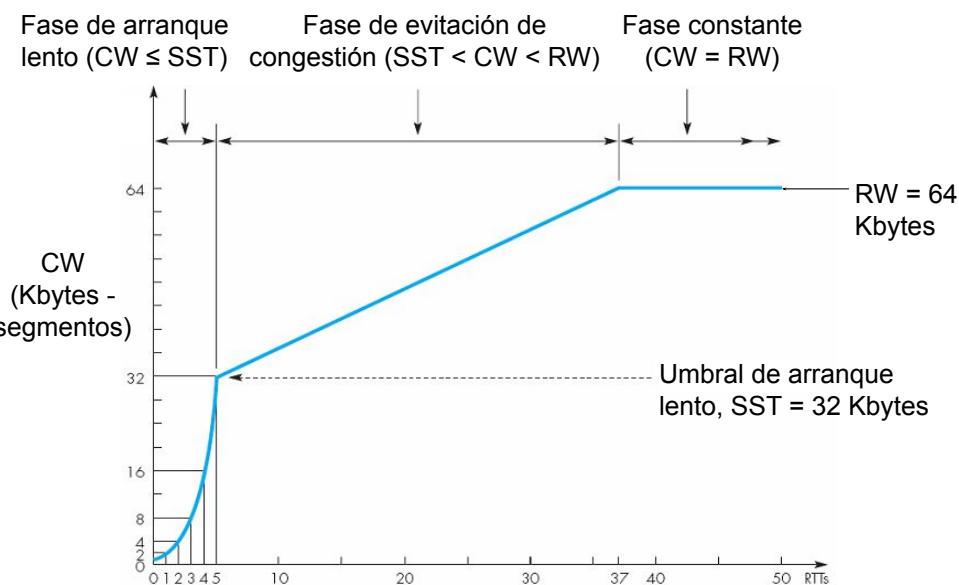
- La ventana de congestión es complementaria a la ventana de recepción (RW) usada para el control de flujo
- En una situación de no congestión (sin pérdida o retraso de segmentos) la ventana de congestión alcanza el mismo tamaño que la ventana de recepción ( $CW = RW$ )
- Cuando se produce una situación de congestión el tamaño de CW se va reduciendo progresivamente
- Cuando la situación de congestión desaparece, el tamaño de CW se va aumentando progresivamente
- El número máximo de bytes que puede enviar el emisor (AW, Allowed Window) es el mínimo de ambos tamaños de ventana:

$$AW = \min \{ RW, CW \}$$

## Control de la Congestión

- La red está sin congestión cuando no se pierden o retrasan segmentos
- La transmisión comienza con un tamaño de ventana de congestión CW = 1
  - El emisor envía un único segmento de tamaño máximo igual a MSS
- A continuación, la CW va aumentando, pasando por tres fases distintas:
  - **Fase de arranque lento (slow start)**
    - La CW se incrementa en uno por cada segmento enviado y confirmado
    - Esto provoca un crecimiento exponencial (CW = 1, 2, 4, 8, 16, 32...)
    - Esta fase termina cuando el tamaño de CW alcanza un cierto umbral, denominado umbral de arranque lento (SST, *Slow Start Threshold*)
    - Inicialmente, el valor del SST suele ser de 64 Kbytes
  - **Fase de evitación de congestión (congestion avoidance)**
    - A partir del SST, la CW se incrementa en uno cada vez que se envía y se confirma una ventana completa (es decir, CW segmentos)
    - Esto provoca un crecimiento lineal
    - Esta fase termina cuando la CW alcanza el tamaño de la ventana de recepción (RW)
  - **Fase constante**
    - En esta fase, la CW se mantiene a un valor constante (CW = RW)

## Control de la Congestión

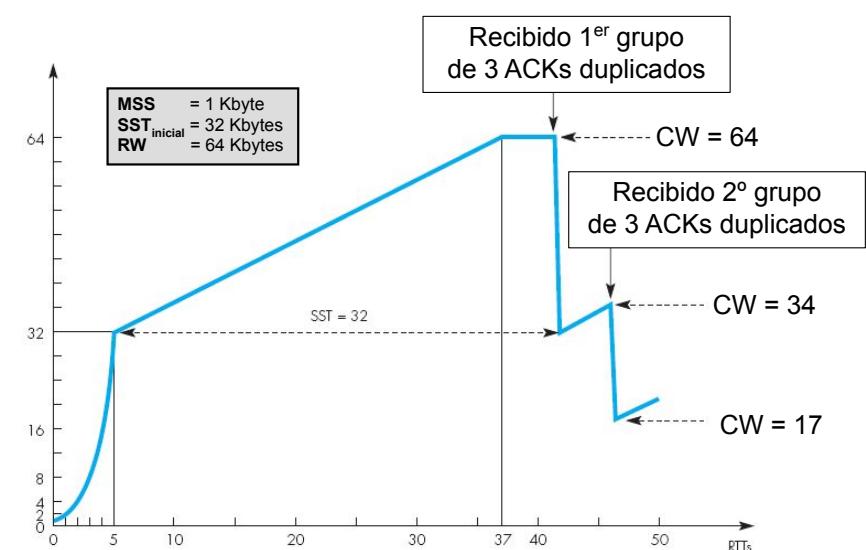


## Control de la Congestión

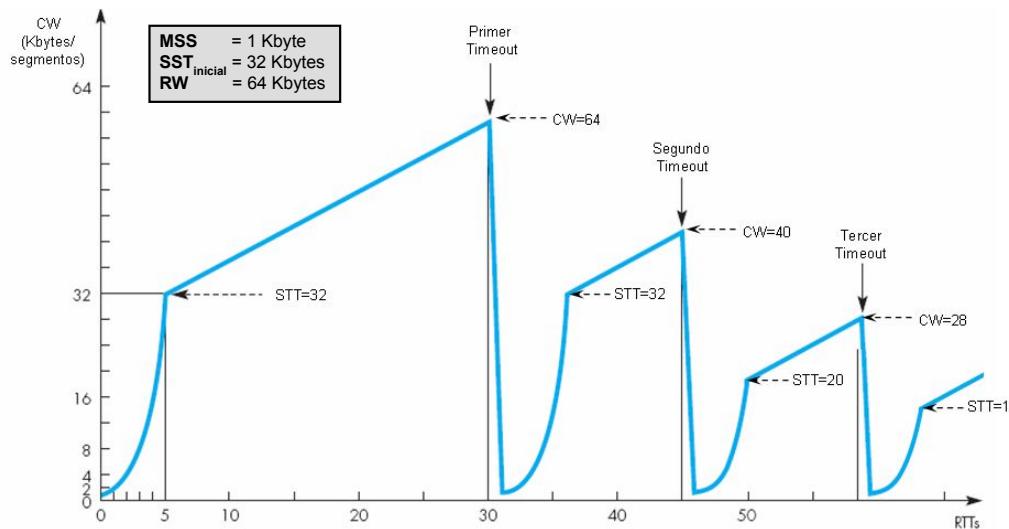
- La situación de congestión en la red se detecta indirectamente
- **Recepción de 3 ACKs duplicados**
  - Nivel de congestión leve, sigue habiendo tráfico en la red (llegan las confirmaciones)
  - Se activa el método de recuperación rápida (*fast recovery*):
    - Se reduce el valor de CW y SST a la mitad del valor de CW
    - Se ejecuta el método de evitación de congestión
- **Espiración del temporizador de retransmisión (RTO)**
  - Nivel de congestión elevado, el tráfico en la red está interrumpido (no llegan confirmaciones)
  - En este caso se realizan las siguientes acciones:
    - Se reduce el valor de SST a la mitad del valor de CW
    - Se inicializa el tamaño de CW a 1
    - Se ejecuta el método de arranque lento

## Control de la Congestión

- **Recepción de 3 ACKs duplicados**



- Expiración del temporizador de retransmisión



Cuando se recibe un segmento TCP con un número de secuencia que ya se ha recibido previamente...

- Se puede retrasar la confirmación del número de secuencia recibido.
- Se confirma inmediatamente el número de secuencia esperado.
- No es necesario confirmar nada

El estado TIME-WAIT del protocolo TCP sirve para...

- Implementar el cierre ordenado de tres vías (3-way)
- Impedir una nueva conexión con los mismos parámetros
- Poder retransmitir los últimos datos enviados antes del cierre

Cuando se recibe un segmento TCP con un número de secuencia mayor al esperado...

- Se puede retrasar la confirmación del número de secuencia recibido.
- No es necesario confirmar nada.
- Se confirma inmediatamente el número de secuencia esperado.

## Ejercicios: Preguntas Teóricas

¿Qué se hace al recibir 3 ACKs duplicados?

- Se envía el primer segmento sin confirmar de la ventana.
- Se ignoran al ser duplicados, es decir, no se hace nada.
- Se espera a que expire el temporizador de retransmisión y se envía el primer segmento sin confirmar de la ventana.



- Un cortafuegos es un componente de seguridad que analiza el tráfico de red y determina si debe permitir su paso. Funciones:
  - Filtrado de paquetes de red
  - Registro de actividad
  - Traducción de direcciones
- **Tipos de cortafuegos:**
  - En función del estado (*stateless/stateful*): Si consideran únicamente las características de los paquetes individuales o si además consideran el estado de la conexión
  - En función de la capa (de red o de aplicación): Si comprueban las cabeceras de los protocolos de red de los paquetes (IP, ICMP, TCP o UDP) o si también consideran sus datos que pertenecen a protocolos de aplicación (ej. HTTP)
- **Filtrado de paquetes** (Netfilter/iptables)
  - Permite manipular reglas asociadas a tablas
  - Las tablas son una funcionalidad ofrecida por el núcleo del SO
  - Incluye un programa en el espacio de usuario para la gestión

### TEMA 1.3. Servicios de Red: Cortafuegos y NAT

**PROFESORES:**

Rubén Santiago Montero  
Eduardo Huedo Cuesta  
Rafael Rodríguez Sánchez

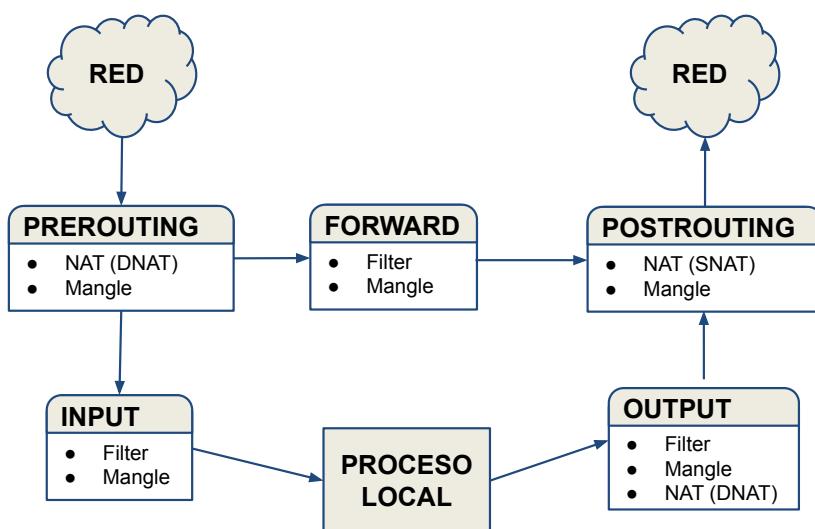
### iptables: Tablas, cadenas y reglas

- **Reglas:** definen qué hacer (ej. descartar o aceptar) con un paquete que cumple unos determinados criterios (ej. puerto origen, dirección IP destino...)
- **Cadenas:** listas de reglas que se aplican en orden a los paquetes en un punto determinado de su procesamiento
  - Una regla puede mover un paquete a otra cadena
  - Todo paquete de entrada/salida del sistema atraviesa al menos una cadena
  - Si un paquete no encaja en ninguna de las reglas, se aplica la política de la cadena
- **Tablas:** conjuntos de cadenas destinados a diferentes tipos de procesamiento

### iptables: Tablas y cadenas predefinidas

- **Tabla Filter**
  - Bloquea o permite el tránsito de un paquete
  - Todo paquete del sistema atraviesa esta tabla
    - Cadena INPUT: paquetes destinados al sistema
    - Cadena OUTPUT: paquetes generados en el sistema
    - Cadena FORWARD: paquetes que atraviesan el sistema (encaminados)
- **Tabla NAT**
  - Re-escribe las direcciones origen/destino y puertos de un paquete
    - Cadena PREROUTING: paquetes de entrada antes de la decisión de encaminamiento
      - Usada en DNAT (Destination NAT)
    - Cadena POSTROUTING: paquetes de salida después de la decisión de encaminamiento
      - Usada en SNAT (Source NAT)
    - Cadena OUTPUT: paquetes de salida generados localmente
- **Tabla Mangle**
  - Sirve para cambiar algunos campos de un paquete (ej. TOS/DS o MSS)
  - Tiene las 5 cadenas anteriores

## iptables: Tablas y cadenas predefinidas



Versión simplificada de cadenas y tablas

## iptables: Definición de Reglas

- Definición de reglas según el estado de la conexión:

Opción	Significado
-m state --state NEW	Filtrado de paquetes correspondientes a conexiones nuevas (el primer paquete)
-m state --state ESTABLISHED	Filtrado de paquetes correspondientes a conexiones ya establecidas
-m state --state RELATED	Filtrado de paquetes relacionados con otras conexiones existentes
-m state --state INVALID	Filtrado de paquetes que no pertenecen a ninguno de los estados anteriores

- Objetivo (target) de una regla (-j, jump)
  - DROP
  - ACCEPT
  - REJECT, como DROP pero envía un mensaje ICMP (--reject-with define el tipo, ej. connection-administratively-filtered, icmp-port-unreachable)
  - LOG

## iptables: Definición de Reglas

- Las reglas se pueden definir según la información del paquete o el estado de la conexión
- Se debe indicar la cadena a la que se añade la regla
- Debe incluir un **objetivo** (*rule target*)

Opción/Ejemplo	Significado
-A INPUT	Añade regla a cadena de entrada
-A OUTPUT	Añade regla a cadena de salida
-A FORWARD	Añade regla a la cadena forward (sólo en caso de routers)
-s 192.168.1.1	Filtrado por dirección IP origen
-d 140.10.15.1	Filtrado por dirección IP destino
-p tcp	Filtrado de paquetes TCP
-p udp	Filtrado de paquetes UDP
-p icmp	Filtrado de paquetes ICMP
--sport 3000	Filtrado por nº de puerto origen (para TCP o UDP)
--dport 80	Filtrado por nº de puerto destino (para TCP o UDP)
--icmp_type 8	Filtrado por tipo de mensaje (para ICMP)
-i eth0	Filtrado por interfaz de red de entrada
-o eth1	Filtrado por interfaz de red de salida

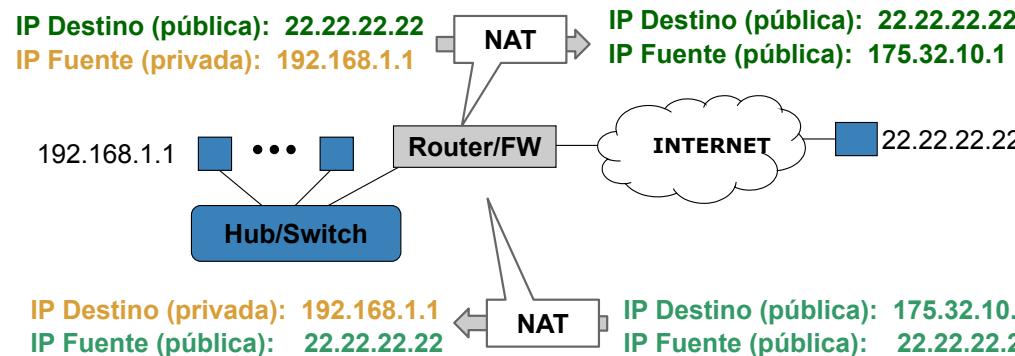
## iptables: Ejemplos de Reglas

```
# Política por defecto para cadenas INPUT, OUTPUT y FORWARD
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP
# Dejar entrar o salir cualquier paquete correspondiente a
# conexiones establecidas o relacionadas
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
# Permitir conexiones entrantes SSH (tcp/22) desde pc-oficina
iptables -A INPUT -s 200.1.1.1 -p tcp --dport 22 -m state \
--state NEW -j ACCEPT
# Permitir conexiones web salientes (tcp/80) a cualquier destino
iptables -A OUTPUT -p tcp --dport 80 -m state --state NEW -j ACCEPT
# Permitir conexiones pop3 salientes (tcp/110) con servidor de correo
iptables -A OUTPUT -d 22.1.1.1 -p tcp --dport 110 -m state \
--state NEW -j ACCEPT
# Permitir conexiones DNS salientes (udp/53) con servidor DNS
iptables -A OUTPUT -d 22.1.1.2 -p udp --dport 53 -m state \
--state NEW -j ACCEPT
```

# NAT: Network Address Translation

## Redes Privadas IPv4

- Permite aliviar el problema del número limitado de direcciones IPv4
- El objetivo es dar acceso a Internet a máquinas en redes privadas



# NAT: Traducción Estática

- Asignación de N direcciones privadas a N direcciones públicas
- Asignación fija
- Ejemplo de tabla de traducción estática para N=7

IP Privada	IP Pública
192.168.1.3	147.96.80.132
192.168.1.23	147.96.80.12
192.168.1.2	147.96.80.122
192.168.1.5	147.96.81.2
192.168.1.4	147.96.81.23
192.168.1.7	147.96.81.77
192.168.1.56	147.96.81.4

# NAT: Traducción Dinámica

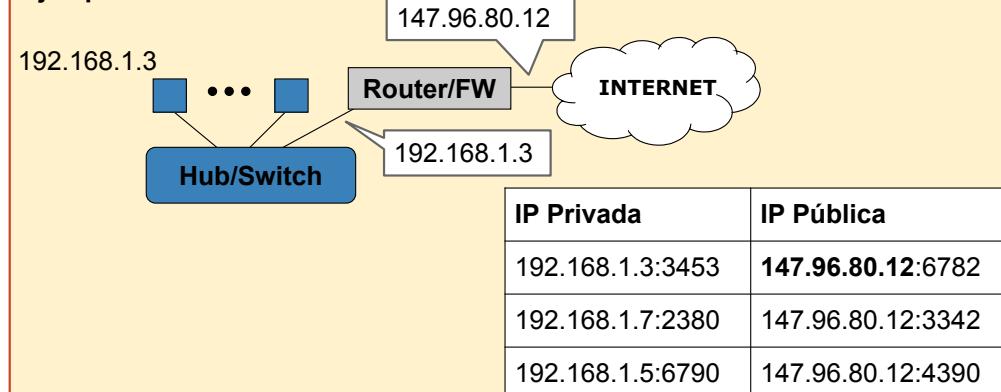
- Asignación de N direcciones privadas a M direcciones públicas ( $M < N$ )
- Asignación dinámica, sólo pueden acceder a Internet M máquinas a la vez
- Ejemplo de tabla de traducción dinámica para N=7, M=3

IP Privada	IP Pública
192.168.1.3	147.96.80.132
192.168.1.23	147.96.80.12
192.168.1.2	147.96.80.122
192.168.1.5	Sin posibilidad de acceso a Internet hasta que se libere una IP pública
192.168.1.4	
192.168.1.7	
192.168.1.56	

# NAT: NAPT - Masquerading

- NAPT (Network Address and Port Translation)
- Asignación de N direcciones privadas a 1 dirección pública
- **Funcionamiento:**
  - La única dirección IP pública disponible es la dirección IP pública del Router
  - El nº de puerto origen del cliente se traduce a un puerto libre del Router

## Ejemplo:



## NAT: NAPT - Masquerading

- El objetivo SNAT de la tabla NAT permite cambiar la dirección origen
- Se aplica a la cadena POSTROUTING
- El resultado se aplica a todos los paquetes posteriores de la conexión
- Permite implementar NAPT con dirección IP pública fija

```
iptables -t nat -A POSTROUTING -o ppp0 -j SNAT --to 175.20.12.1
```

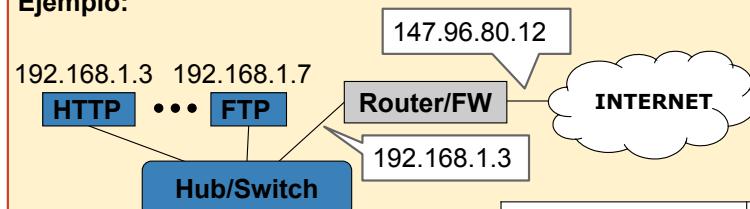
- El objetivo MASQUERADE de la tabla NAT permite usar una dirección IP pública dinámica
  - Usa la dirección IP del interfaz como dirección IP origen, que puede cambiar de una conexión a otra, al ser dinámica
  - Además, mantiene pista de las conexiones activas para aplicar también el cambio

```
iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE
```

## NAT: Port Forwarding - Virtual Servers

- Asignación de **1 dirección pública** a N direcciones privadas
- Permite tener servidores en la red privada “visibles” desde Internet
- **Funcionamiento:**
  - Desde Internet, todos los servidores usan la dirección IP pública del Router
  - El Router traduce y reenvía los paquetes al servidor real de la red interna

### Ejemplo:



IP Privada	IP Pública
192.168.1.3:8080	147.96.80.12:80
192.168.1.7:20	147.96.80.12:20
192.168.1.7:21	147.96.80.12:21

## NAT: Port Forwarding - Virtual Servers

- El objetivo DNAT de la tabla NAT permite modificar la dirección de destino de un paquete y, opcionalmente, el puerto
- Se aplica a las cadenas OUTPUT y PREROUTING
- El resultado se aplica a todos los paquetes posteriores de la conexión

```
iptables -t nat -A PREROUTING -d 175.20.12.1 -p tcp --dport 80 \
-j DNAT --to 192.168.1.1:8080

iptables -t nat -A PREROUTING -d 175.20.12.1 -p tcp --dport 25 \
-j DNAT --to 192.168.1.2

iptables -t nat -A PREROUTING -d 175.20.12.1 -p tcp --dport 20 \
-j DNAT --to 192.168.1.7

iptables -t nat -A PREROUTING -d 175.20.12.1 -p tcp --dport 21 \
-j DNAT --to 192.168.1.7
```

## Ejercicios: Preguntas Teóricas

La tabla filter de iptables sirve para...

- Cambiar algunos campos de un paquete, como TOS o TTL.
- Bloquear o permitir el tránsito de un paquete.
- Reescribir las direcciones origen/destino de un paquete.

La tabla nat de iptables sirve para...

- Cambiar algunos campos de un paquete, como TOS o TTL.
- Bloquear o permitir el tránsito de un paquete.
- Reescribir las direcciones y puertos origen/destino de un paquete.



# Domain Name System (DNS)

- Mantiene, entre otras cosas, la asignación entre nombres de dominio y direcciones IP
- DNS está implementado como una BD distribuida:
  - Cada sitio guarda información únicamente de sus sistemas
  - Se intercambia y comparte la información con otros sitios
  - DNS recibe y realiza consultas sobre los nombres de dominio
- DNS es un sistema muy complejo:
  - Definido en aproximadamente 108 RFCs
  - Múltiples implementaciones con diferente funcionalidad, por ejemplo:
    - BIND (el más usado)
    - Microsoft DNS, djbdns, NSD, Unbound, PowerDNS
- DNS define:
  - Un espacio de nombres jerárquico de nombres de dominio y direcciones IP
  - Una BD distribuida
  - Un mecanismo para encontrar servicios de red
  - Un protocolo para intercambiar información
  - Herramientas cliente (*resolvers*) para consultar la BD

## TEMA 1.3. Servicios de Red: DNS

### PROFESORES:

Rubén Santiago Montero  
 Eduardo Huedo Cuesta  
 Rafael Rodríguez Sánchez

## Zonas y Dominios

### Dominio raíz

- Contienen referencias a los servidores de nombres de los dominios de 1<sup>er</sup> nivel
- 13 servidores de nombres [a-m].root-servers.net (múltiples máquinas - *anycast*)

### Top Level Domains (TLDs)

- Gestionados por ICANN
- Lista completa en <http://www.iana.org/domains/root/db>
- Cada zona incluye los servidores de nombres autorizados y los servidores de nombres de los subdominios delegados

### Generic (gTLD)

### Country code (ccTLD)

com	gov	net	edu	...	org	uk	eu	fi	...	es	( <a href="http://www.dominios.es">www.dominios.es</a> )
-----	-----	-----	-----	-----	-----	----	----	----	-----	----	--



### Dominio

- Subárbol del espacio de nombres de dominio
- Gestión delegada en varias organizaciones

### Zona

- Una organización de gestión
- Contiene información de la zona y servidores de nombres de subdominios delegados

## Nombres de Dominio

### Nombre de dominio completo (FQDN, Fully Qualified Domain Name)

- Lista de nombres de nodo o etiquetas de dominio (ej. www, printer-server...) que representan la jerarquía desde el nivel más bajo hasta el raíz (aunque se suele omitir), utilizando el carácter de punto como separador entre etiquetas
  - Ejemplo: www.ucm.es. (parte más significativa, “.”, a la derecha)

### Espacio de nombres para direcciones IP

- Para búsqueda inversa: obtener el nombre de dominio asociado a una IP
- La dirección IP se invierte para que la parte más significativa esté a la derecha
- Para IPv4 se usa el dominio in-addr.arpa.
  - Ejemplo: 63.173.189.1 → 1.189.173.63.in-addr.arpa.

### Restricciones en los nombres de dominios

- No hay límite en el número de subdominios de la jerarquía
- El FQDN puede ocupar un máximo de 256 caracteres (incluyendo los puntos)
- Cada sección del FQDN puede tener un máximo de 63 caracteres
- No diferencia entre mayúsculas y minúsculas
- Formados por caracteres alfanuméricos y guiones

## Funcionamiento: Registros

- La BD de DNS se estructura en registros (*Resource Records, RR*)
  - DNS gestiona diferentes tipos de registros para almacenar servidores de nombres, asignaciones nombre-IP e IP-nombre, servidores de mail...
  - Los registros son estándar e independientes de la implementación
  - Son la información básica que se intercambia y almacena en los servidores
- Los servidores guardan los registros de sus dominios en ficheros de zona (*zone file*) en formato de texto
  - Ejemplo: piscis.midominiodns.com ↔ 147.96.80.1



## Funcionamiento: Protocolo DNS

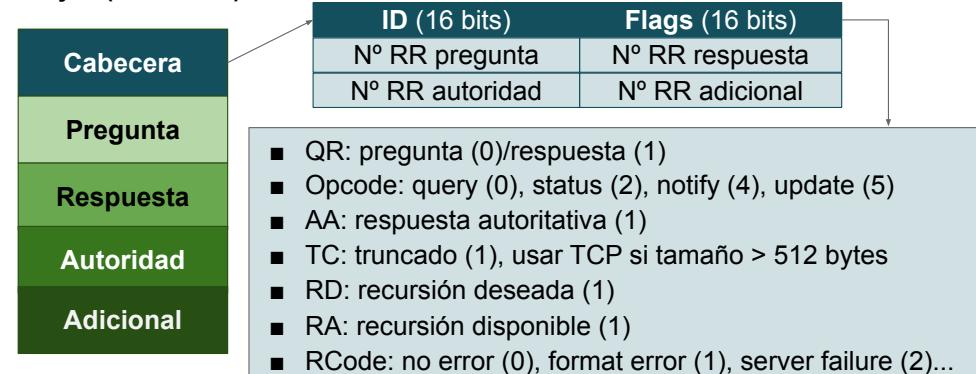
```
;; Got answer:  
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19305  
;; flags: qr rd; QUERY: 1, ANSWER: 0, AUTHORITY: 7, ADDITIONAL: 14  
;; WARNING: recursion requested but not available  
  
;; QUESTION SECTION:  
;informatica.ucm.es. IN A  
  
;; AUTHORITY SECTION:  
es. 172800 IN NS f.nic.es.  
es. 172800 IN NS g.nic.es.  
es. 172800 IN NS a.nic.es.  
...  
  
;; ADDITIONAL SECTION:  
a.nic.es. 172800 IN A 194.69.254.1  
a.nic.es. 172800 IN AAAA 2001:67c:21cc:2000::64:41  
...
```

## Funcionamiento: Protocolo DNS

### Protocolo de Transporte

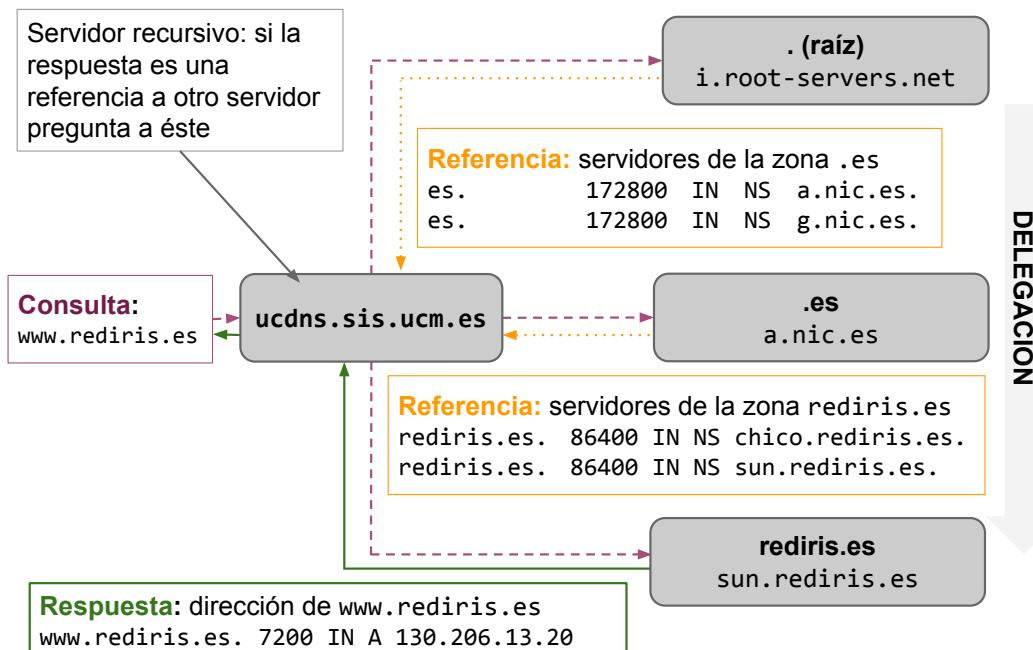
- Principalmente, UDP en el puerto 53
- TCP para transferencias de zona o respuestas de más 512 bytes (RFC 5966)

### Mensajes (RFC 1035)



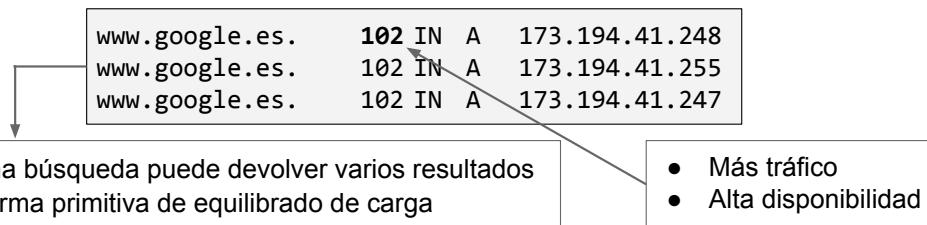
- La sección Pregunta (en preguntas y respuestas) incluye el nombre de dominio y el tipo de registro por el que se pregunta
- La sección Autoridad especifica los servidores autoritativos de los dominios
- La sección Adicional incluye registros que pueden ser de ayuda (*resolver*)

## Funcionamiento: Delegación y Resolución



## Funcionamiento: Caching

- Almacenar la resolución de direcciones mejora notablemente la eficiencia
  - La relación nombre-IP es prácticamente estática
- Las respuestas se almacenan durante un TTL (“time-to-live”), que varía para cada entrada según su probabilidad de cambio:
  - Servidores de “.es”, 2 días (172800)
  - Servidores de “.rediris.es”, 1 día (86400)
  - IP de www.rediris.es, 2 horas (7200)
- Los clientes y servidores de *cache* pueden observar o no el TTL
- *Cache negativa*, cuando una búsqueda falla:
  - Ningún dominio encaja en el nombre buscado
  - El registro solicitado no existe para el recurso
  - El servidor no responde o no se puede alcanzar por problemas de red



## Servidores de Nombres

### Autoritativos (*authoritative*)

- Representan oficialmente a la zona
- Primario o maestro: tiene la copia oficial en disco de la BD
- Secundarios o esclavos: obtienen la BD de los primarios (*zone transfer*)
- La especificación de DNS establece que debe haber un servidor primario y al menos uno secundario por zona

### De *cache* (*caching-only*)

- Guardan los resultados de las búsquedas realizadas partiendo de una lista de servidores del dominio raíz
- No tienen ningún registro DNS propio, ni son autoritativos para ninguna zona
- Reducen la latencia de las consultas y el tráfico DNS en la red

### Recursivos y no-recursivos

- No-recursivos: cuando no disponen el registro de la consulta, devuelven una referencia al servidor de nombres que puede tenerlo
- Recursivos: resuelven cada referencia hasta devolver la respuesta al cliente
- Los servidores autoritativos suelen ser no-recursivos
- En la configuración de los clientes deben usarse servidores recursivos

## La Base de Datos de DNS

- Archivos de texto (*zone files*) mantenidos en el servidor primario de la zona
- **Directivas**, que especifican cómo interpretar los registros. Directivas estándar:
  - \$ORIGIN: dominio por defecto que se añade a todos los nombres que no sean FQDN
  - \$INCLUDE: incluye un archivo con registros, permite mantener separados los registros de datos en diferentes archivos
  - \$TTL: valor por defecto para el TTL de los registros
- **Registros de Recursos (RR)**, que se asocian a la zona

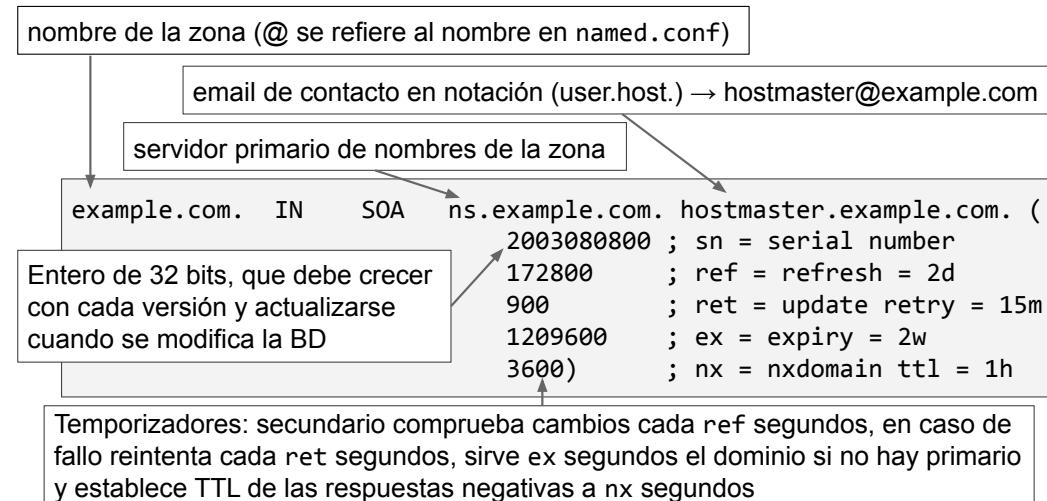
### Formato de los registros (RFCs 1034 y 2181)

[nombre] [ttl] [clase] tipo datos

- nombre: que identifica el registro, normalmente nombre de host o dominio
- ttl: tiempo en segundos que se puede almacenar y considerarse válido
- clase: normalmente IN (Internet)
- tipo: Clasificados en 4 grupos (Zona, Básicos, Seguridad y Opcionales), hay gran cantidad de tipos aunque sólo unos pocos se usan habitualmente
- datos: Depende del tipo de registro

## La Base de Datos de DNS: Registro SOA

- El registro Start of Authority (**SOA**) marca el comienzo de definición de una zona
- La zona incluye los registros dentro del espacio de nombres DNS
- Un servidor DNS tiene normalmente dos zonas:
  - Zona directa (*forward*): traducción nombre → IP
  - Zona inversa (*reverse*): traducción IP → nombre



## La Base de Datos de DNS: Registro NS

- El registro Name Server (**NS**) especifica los servidores autoritativos para la zona
- Además se incluyen los servidores de nombres de los subdominios delegados a otras organizaciones
- Normalmente se añaden después del registro SOA (puede omitirse el nombre por ser el mismo)

Nombre del registro anterior (example.com en SOA)

sub	NS ns.example.com.	NS ns1.example.com.	NS ns-ha.example.com.	NS ns.sub.example.com.	NS ns.example.com.
-----	--------------------	---------------------	-----------------------	------------------------	--------------------

“.” al final para los FQDN

Se incluyen los subdominios para que funcione la delegación, aunque la información corresponde a la zona del subdominio (*glue records*)  
Ejemplo: NS de .com debe incluir los NS listados en esta zona (example.com)

## La Base de Datos de DNS: Registros A y PTR

### Registro A y AAAA

- El registro Address (**A** para IPv4 y **AAAA** para IPv6) es la base de DNS
- Incluye la traducción directa (nombre → IP)

ns	IN A	63.175.177.1
ns1.example.com.	IN A	63.175.177.4
ns1.example.com.	IN AAAA	2001:501:2f::a01b
ns1.example.com.	IN A	63.175.177.2

No es FQDN, por lo que se completa con \$ORIGIN.  
Hay múltiples registros para ns.example.com.

### Registro PTR

- El registro Pointer (**PTR**) contiene la traducción inversa (IP → nombre)
- Se organizan en diferentes zonas para cada subred (o redefiniendo \$ORIGIN)

1.177	IN PTR	ns.example.com.
-------	--------	-----------------

Relativo a 175.63.in-addr.arpa

Usar FQDN para que no añada \$ORIGIN

## La Base de Datos de DNS: Registro MX

- El registro Mail eXchanger (**MX**) es usado por los sistemas de correo para encaminar los mensajes eficientemente
- Permite recibir de forma centralizada el correo de una organización y realizar operaciones centralizadas (ej. filtrar spam)

Prioridad, valores menores son más prioritarios

example.com.	IN MX	10 mail
example.com.	IN MX	20 mail2.example.com.

MTA con e-mail a [usuario@example.com](mailto:usuario@example.com) usará  
mail.example.com (más prioritario)

## La Base de Datos de DNS: Registro CNAME

- El registro Canonical Name (**CNAME**) se usa para definir el nombre canónico de un nombre de dominio, lo que permite definir un *alias* para el nombre canónico
- Deben siempre apuntar a un dominio (nunca a una IP)
- Un *alias* definido por un CNAME no debe tener otros registros
- MX y NS no pueden apuntar a un CNAME

Este es el nombre canónico

informatica.ucm.es.	86400	IN CNAME	ucm.es.
ucm.es.	86400	IN A	147.96.1.15

- informatica.ucm.es. es un alias de ucm.es.
- También se resuelve el nombre canónico

## La Base de Datos de DNS: Ejemplo

```
; Ejemplo para la zona example.com
$TTL 2d ; TTL por defecto = 2 días o 172800 segundos
$ORIGIN example.com.

example.com. IN SOA ns.example.com. admin.example.com. (
    2003080800 ; serial number (año,mes,día,secuencia)
    3h          ; refresh = 3 horas
    15M         ; update retry = 15 minutos
    3W12h       ; expiry = 3 semanas + 12 horas
    2h20M)      ; nx ttl = 2 horas + 20 minutos

    IN NS ns
    IN NS ns-backup
    IN MX 10 mail ; equivale a mail.example.com.
    IN MX 20 mail2.example.com. ; servidor de respaldo

; todos los servidores necesitan un registro A
ns      IN A 192.168.0.10
ns-backup IN A 192.168.0.11
mail    IN A 192.168.0.12
mail2   IN A 192.168.0.13
www    IN A 192.168.0.50
```

## BIND

- Berkeley Internet Name Domain (BIND) es una implementación *open source* del protocolo DNS
- Las versiones comunes son BIND9 y BIND10
- Componentes:
  - Servidor de nombres: named
  - Programa de gestión remota del servidor: rndc
  - Clientes: dig, nslookup and host
  - Librerías clientes asociadas para la consulta de servidores DNS
- **Archivos de configuración:**
  - named.conf, que especifica las configuración del servidor (tipo, control de acceso...)
  - Archivos de texto con la BD de la zona

## Ejercicios: Preguntas Teóricas

Respecto a las direcciones IP, el servicio de nombres de dominio (DNS)...

- Establece un dominio específico para buscar su nombre de dominio asociado.
- DNS solo maneja la traducción de nombres de dominio en IP.
- Solo permite las búsquedas inversas para los dominios TLD (*top level domains*).

Los servidores autoritativos de una base de datos DNS...

- Guardan los resultados de las búsquedas realizadas.
- Suelen ser recursivos.
- Representan oficialmente a la zona.



## TEMA 1.4. Protocolo IPv6

**PROFESORES:**

Rubén Santiago Montero  
Eduardo Huedo Cuesta  
Rafael Rodríguez Sánchez

# Introducción: Limitaciones IPv4

- Direccionalamiento muy limitado
  - Direcciones de 32 bits ( $4,3 \cdot 10^9$  direcciones)
  - Soluciones parciales:
    - Uso de direcciones sin clases (CIDR)
    - Uso de intranets con direcciones privadas (NAT)
    - Uso de direcciones dinámicas (DHCP)
- Formato complejo de la cabecera del paquete
  - Longitud variable (campo opciones)
  - Información de fragmentación (casi nunca necesaria)
- Seguridad limitada
  - No incluye soporte para seguridad o autenticación
  - Solución: IPsec
- Soporte limitado para prioridad de tráfico o clase de servicio
  - Funcionalidad no implementada en la mayoría de encaminadores
- Multicast limitado
  - No se ha llegado a utilizar de forma completa y eficaz

## Introducción: Características IPv6

- Espacio de direcciones mucho mayor
  - Direcciones de 128 bits ( $3.4 \cdot 10^{38}$  direcciones)
- Formato de cabecera más simple
  - Mayor velocidad de procesamiento en los encaminadores
- Posibilidad de autoconfiguración de interfaces
- Mejor soporte para opciones adicionales
  - Las opciones de IPv6 no se codifican en la cabecera, sino en el cuerpo del paquete IP mediante cabeceras de extensión
  - Dispone de mayor espacio para su codificación
  - Permite introducir nuevas opciones en el futuro
- Opciones de seguridad tanto para autenticación como para cifrado
- Soporte para tráfico en tiempo real (ej. VoIP)
- Encaminamiento jerárquico basado en prefijos
- Mecanismos de transición desde la versión 4

## Introducción: IPv4 e IPv6

Característica	IPv4	IPv6
Longitud de direcciones	32 bits	128 bits
Clases de direcciones	Clases A, B y C o CIDR	Direcciones sin clase
Tipo de direcciones	Unicast, Multicast, Broadcast	Unicast, Multicast, Anycast
Configuración de dirección	Estática (a través de ficheros de configuración) o por DHCP	Estática (a través de ficheros de configuración), autoconfiguración (plug and play) o por DHCP
Formato cabecera	Complejo. Longitud variable	Simple. Longitud fija
Calidad de servicio	Sí, aunque no soportado totalmente	Sí
Soporte tráfico en tiempo real	No	Sí
Seguridad	No (extensión IPsec)	Sí



## Direccionamiento

# Direcciones IPv6: Tipos de Direccionamiento

### Unicast

- Identifican a un único interfaz en la red
- Un paquete dirigido a una dirección unicast se entregará únicamente al interfaz identificado con dicha dirección IP

n	128 - n
Prefijo de la subred	Identificador del interfaz

### Multicast

- Identifican a un grupo de interfaces (asignadas a más de un interfaz)
- Un paquete dirigido a una dirección multicast se entrega a **todos** los interfaces identificados con esa dirección
- No existe dirección de broadcast

### Anycast (RFC 2461 y RFC 1884)

- Identifican a un grupo de interfaces (asignadas a más de un interfaz)
- Un paquete dirigido a una dirección anycast se entrega a **uno solo** de los interfaces identificados con esa dirección, normalmente al más cercano, en función de la métrica usada por el protocolo de encaminamiento
- Se asignan del espacio de direcciones unicast

## Direcciones IPv6: Notación

- Las direcciones tienen una longitud de 128 bits (16 bytes)
- Notación hexadecimal
  - La dirección se divide en 8 grupos de 16 bits
  - Cada grupo se escribe en hexadecimal con 4 dígitos
  - Los grupos se separan con ":"  
`FDEC:BA98:7654:3210:0123:4567:89AB:CDEF`  
`FE80:0000:0000:0000:0008:0800:200C:741A`
- Notación abreviada
  - En cada grupo los ceros a la izquierda se pueden omitir  
`0000 → 0`    `0074 → 74`
  - Una cadena de ceros seguidos se puede comprimir con el símbolo ":"  
`FE80:0:0:0:8:800:200C:741A → FE80::8:800:200C:741A`
  - El símbolo ":" sólo puede aparecer una vez (cadena mayor o primera)  
`21AB:0:0:A:0:0:1234:5678 →`  
    → `21AB::A:0:0:1234:5678` (incorrecto, ambiguo)  
    → `21AB::A:0:0:1234:5678` (correcto)

## Direcciones IPv6: Notación CIDR

- Las direcciones IPv6 son sin clase para soportar el direccionamiento jerárquico
- Se dividen en prefijo y sufijo
- La longitud del prefijo se denota en CIDR
- Ejemplos:

`FDEC:BA98:7654:3210:0123:4567:89AB:CDEF/64`

`FE80:0:0:0:8:800:200C:741A/64`

# Direcciones IPv6: Ámbitos (RFC 4007)

- **Ámbito (scope):** Determina en qué parte de la red es válida la dirección
  - **Enlace local (*link-local*):** Válida dentro del enlace en el que está conectado la interfaz de red (ej. una LAN)
  - **Sitio local (*site-local*):** Válida dentro de un *sitio* formado por una o varias redes interconectadas mediante encaminadores (ej. campus universitario)
  - **Global:** Válida en todo Internet
- **Zona (scope zone):** Región conexa de la red de un ámbito determinado
  - Por ejemplo, una zona de enlace local consiste en un enlace y todos los interfaces directamente conectados, y la zona global única comprende todos los interfaces y enlaces de Internet
  - La unicidad de las direcciones sólo se garantiza dentro de su zona
  - Los datagramas no se redirigen a una zona distinta, aunque sea del mismo ámbito
  - En caso de ambigüedad, se debe usar <dirección>%<id\_zona>, por ejemplo, fe80::1234%eth1 para direcciones de enlace local

# Direcciones IPv6: Estructura

- IPv4 tiene una estructura de un nivel (red y host)
- IPv6 permite una jerarquía flexible, que acomoda diferentes tipos de direcciones
- Cada tipo de dirección comienza con un prefijo (prefijo de formato) de longitud variable

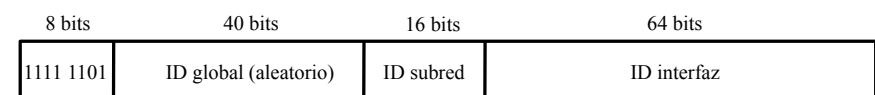
Tipo de dirección	FP (binario)	FP (hexadecimal)
Reserved Address	0000 0000	::/8
Global Unicast Address	001	2000::/3
Link-Local Unicast Address	1111 1110 10	FE80::/10
Site-Local Unicast Address (en desuso)	1111 1110 11	FEC0::/10
Unique Local Address (ULA)	1111 110	FC00::/7
Multicast Address	1111 1111	FF00::/8

## Direcciones IPv6: Enlace local

- Direcciones unicast privadas que se asignan a un enlace (*link*) y nunca se encaminan fuera de la zona de ámbito del enlace
  - Es un espacio de direcciones plano
  - Su principal uso es la autoconfiguración y el descubrimiento de vecinos
- Formato:
  - Prefijo de formato (10 bits): 1111 1110 10 (FE80::/10)
  - Los siguientes 54 bits son 0
  - Identificador de interfaz (64 bits)
- **Ejemplo:**  
`fe80::2e81:58ff:fee9:64bb/64`

## Direcciones IPv6: ULA (Unique Local Address)

- Direcciones unicast privadas, definidas en RFC 4193, para usar en intranets jerárquicas, pero no encaminables en Internet (aunque su ámbito es global)
  - Sustituyen a las direcciones de sitio local, definidas en RFC 3879
  - Permiten la auto-configuración
- Formato:
  - Prefijo de formato (7 bits): fc00::/7
  - Bit 8: 1 indica que el prefijo se asigna localmente (0 no está definido)
  - Identificador global (40 bits): pseudo-aleatorio para evitar colisiones
  - Identificador de subred (16 bits): 65.536 subredes por sitio, para crear la estructura de red interna
  - Identificador de interfaz (64 bits)
- **Ejemplo:**  
`fd12:A128:e8e1:1:FEDC:BA98:7865:4321/64`

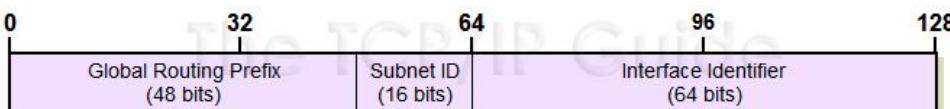


# Direcciones IPv6: Unicast Globales

- Direcciones unicast globales, definidas en RFC 3587, para uso en Internet
  - Permiten la autoconfiguración
- Formato:
  - Prefijo global de encaminamiento (48 bits): Actualmente, IANA está asignando el rango  $2000::/3$ , que permite  $2^{45}$  sitios diferentes. Es la única parte relevante en el encaminamiento global y puede subdividirse jerárquicamente de acuerdo a las necesidades de los RIRs (*Regional Internet Registry*) y LIRs (*Local Internet Registry*)
  - Identificador de subred (16 bits): 65.536 subredes por sitio, para crear la estructura de red interna
  - Identificador de interfaz (64 bits)

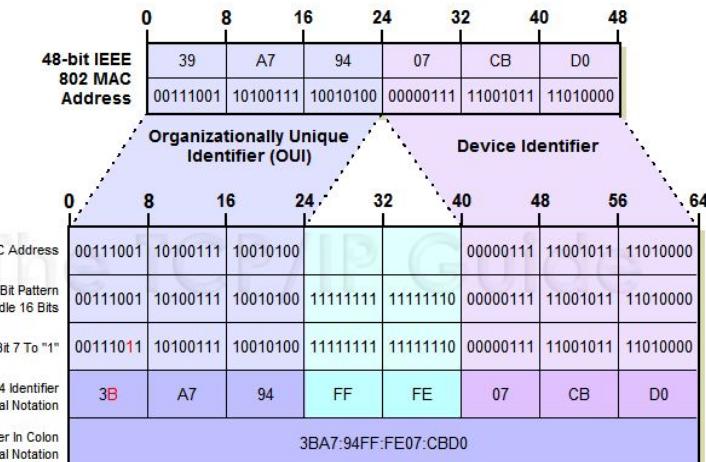
## Ejemplo:

2004:A128::32:FEDC:BA98:7865:4321/64



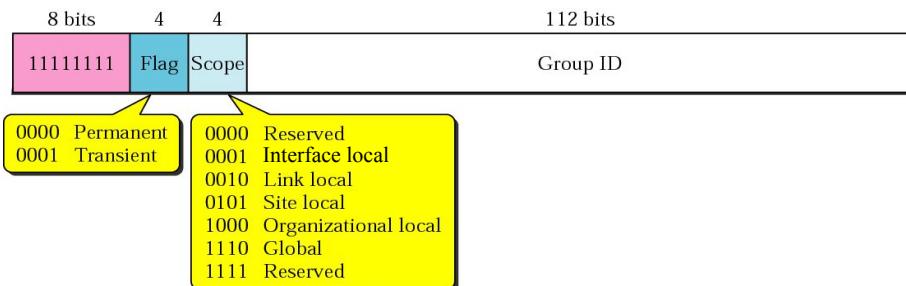
# Direcciones IPv6: ID de Interfaz

- 64 bits menos significativos de la dirección
  - Se genera a partir de la dirección MAC (EUI-48) usando el procedimiento EUI-64 (64-bit Extended Unique Identifier) modificado
  - O, activando las extensiones de privacidad (RFC 4941), se genera de forma pseudoaleatoria y temporal para evitar el rastreo de los clientes



# Direcciones IPv6: Multicast

- Definen un grupo de interfaces en un ámbito determinado
- Formato:
  - Prefijo de formato (8 bits): FF::/8
  - Flags (4 bits): indican si es una dirección permanente (IANA) o temporal para una comunicación (ej. grupo de nodos en una teleconferencia)
  - Ámbito (4 bits)
  - Identificador de grupo (112 bits)
- Las direcciones MAC se generan con el prefijo 33:33: y los 32 bits menos significativos de la dirección IPv6



# Direcciones IPv6: Multicast

- Direcciones para los nodos

Dirección	Ámbito	Significado
FF01::1	<i>Interface local</i>	Un datagrama dirigido a esta dirección se envía sólo al interfaz del nodo
FF02::1	<i>Link local</i>	Un datagrama dirigido a esta dirección se envía a todos los interfaces del enlace local, pero nunca se encamina

- Direcciones para los encaminadores

Dirección	Ámbito	Significado
FF02::2	<i>Link Local</i>	Un datagrama dirigido a esta dirección se envía a todos los encaminadores del enlace.
FF05::2	<i>Site local</i>	Un datagrama dirigido a esta dirección se envía a todos los encaminadores del sitio local, por tanto se reexpide a todas las subredes a través de los encaminadores internos
FF02::9	<i>Link local</i>	Un datagrama dirigido a esta dirección se envía a todos los encaminadores del enlace local que realizan encaminamiento RIP

# Direcciones IPv6: Multicast

- Direcciones multicast de nodo solicitado (*Solicited-node multicast addresses*), usadas en el protocolo de descubrimiento de vecinos
- Se calculan como función de la dirección unicast del nodo:  
**FF02:0:0:0:0:1:FF00::/104 + 24 bits menos significativos de la dirección**
- El rango de direcciones:  
FF02:0:0:0:0:1:FF00:0000 - FF02:0:0:0:0:1:FFFF:FFFF
- Ejemplo:  
Dirección unicast: 2037::01:800:200E:8C6C  
Dirección multicast de nodo solicitado: FF02::1:FF0E:8C6C

# Direcciones IPv6: Otras Direcciones

- Dirección sin especificar: 0:0:0:0:0:0:0:0 (::)
  - Indica que el interfaz no tiene ninguna dirección asignada
- Dirección de *loopback*: 0:0:0:0:0:0:1 (::1)
  - Análoga a la dirección de *loopback* IPv4 (127.0.0.1)
- Direcciones asignadas a IPv4 (*IPv4-mapped*): ::FFFF:<IPv4>
  - De uso en arquitecturas que mezclan las pilas IPv4 e IPv6
  - Ejemplo:  
::FFFF:192.02.13.123 (en notación mixta)

## ¿Cuántas direcciones IPv6 tiene un nodo?

- Dirección de enlace local para cada interfaz
- Direcciones unicast o anycast configuradas para los interfaces
- Dirección de loopback

## Además, responde a las direcciones multicast:

- Dirección multicast de todos los nodos
- Dirección multicast de nodo solicitado para cada dirección configurada
- Direcciones multicast de otros grupos a los que pertenezca el nodo

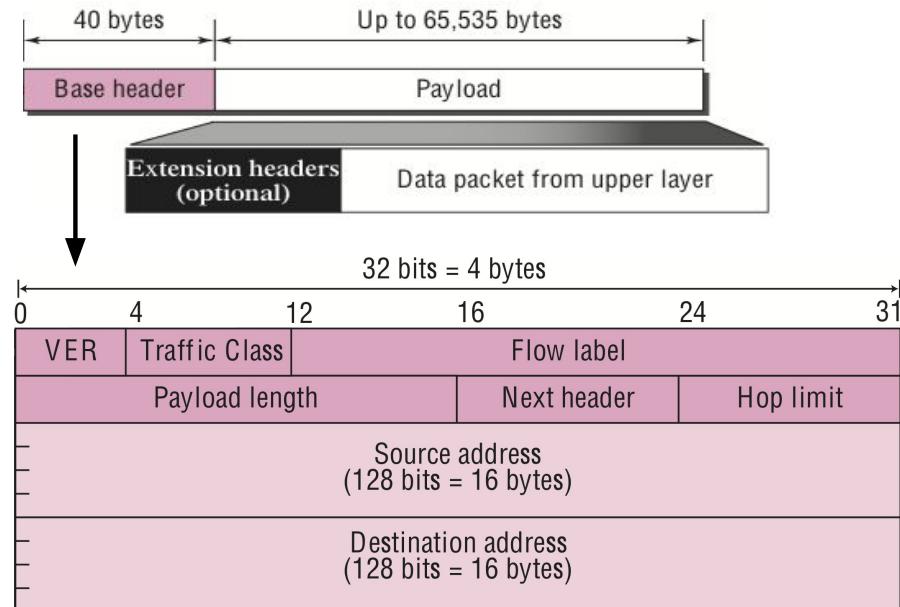


## AMPLIACIÓN DE SISTEMAS OPERATIVOS Y REDES

Grados Ingeniería en Informática  
Universidad Complutense de Madrid

## Datagrama

## Datagrama IPv6: Formato

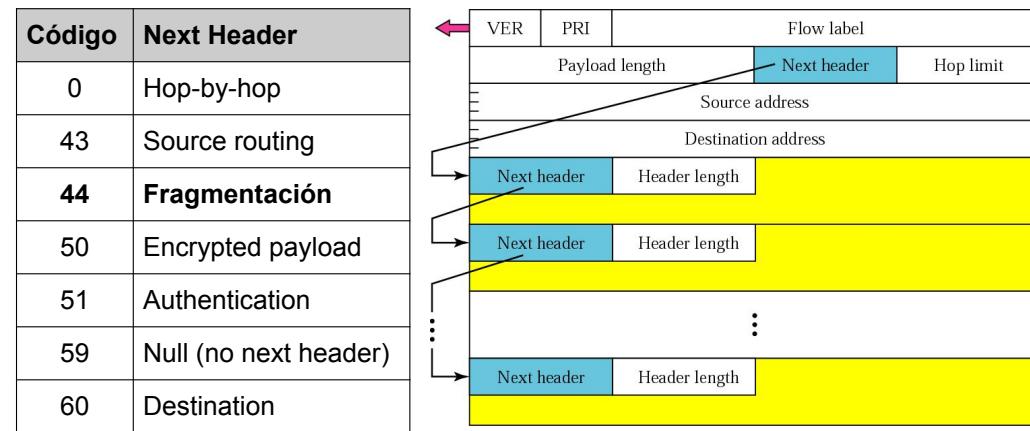


# Datagrama IPv6: Formato

- **Version (4 bits):** 6
- **Traffic Class (8 bits):** distingue diferentes requisitos de entrega del datagrama, es similar al campo DS (antes ToS) de IPv4
  - DSCP (*Differentiated Services Code Point*, 6 bits): Clasificación del tráfico en grupos con distintos requisitos de calidad de servicio
  - ECN (*Explicit Congestion Notification*, 2 bits): Permite detectar situaciones de congestión en la red sin descartar paquetes
- **Flow Label (20 bits):** Etiqueta el paquete como perteneciente a un flujo para mejorar el procesamiento realizado por los encaminadores de la red
  - Un flujo comparte las mismas características (origen/destino, requisitos...)
  - Usado por protocolos de tiempo real y reserva (RTP/RSPV)
- **Payload Length (16 bits):** Longitud sin contar la cabecera (máx. 64 Kbytes)
- **Hop Limit (8 bits):** Similar al campo TTL de IPv4
- **Source/Destination Address (128 bits):** Direcciones origen y destino

# Datagrama IPv6: Formato

- **Next Header (8 bits):** Define la siguiente cabecera en el datagrama, encapsulada en la sección de datos, y puede ser:
  - Una cabecera de extensión, similar al campo opciones de IPv4
  - La cabecera del protocolo de nivel superior (6=TCP, 17=UDP, 58=ICMPv6...)



**Header Length (8 bits):** Longitud de la cabecera en unidades de 8 bytes

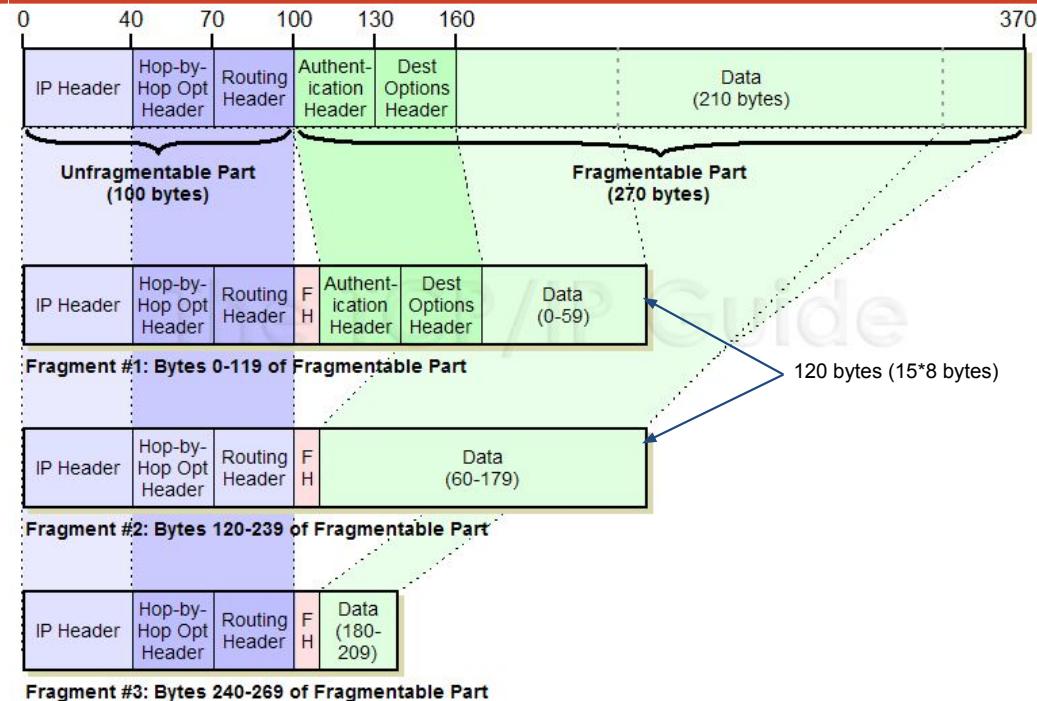
# Datagrama IPv6: Fragmentación

- En IPv6, la fragmentación se realiza en origen (nunca en los encaminadores)
- Se recomienda que los nodos IPv6 implementen el algoritmo Path MTU Discovery (RFC 1981), para descubrir y aprovechar MTUs de camino (la mínima MTU de todos los enlaces del camino) mayores de 1280 bytes (MTU mínima en IPv6)
- Se espera que el protocolo de nivel superior limite el tamaño de los datos según la MTU del camino. No obstante, si no es capaz de hacerlo y genera un paquete mayor que la MTU del camino, el emisor lo dividirá en fragmentos
- **Formato** de la cabecera de fragmentación:

Next header	Header length	Fragmentation offset	0	M
Fragment identification				

- **Header Length (8 bits):** Reservado, inicializado a 0s
- **Offset (13 bits):** desplazamiento respecto al inicio de la parte fragmentable del datagrama original en unidades de 8-bytes (el desplazamiento del primer fragmento es 0)
- **Flags:** M (*More fragments*) indica si hay más fragmentos o no
- **Identification:** permite identificar a los fragmentos del mismo datagrama

# Datagrama IPv6: Fragmentación



# Datagrama IPv6: Comparativa IPv4

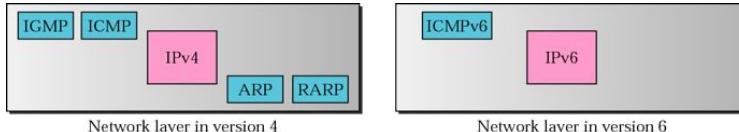


- El campo Header Length se ha eliminado, ya que la longitud es fija
- El campo ToS se ha eliminado y sustituido por el campo Traffic Class (en IPv4, se sustituyó por el campo DS)
- Se ha añadido el campo Flow Label
- El tamaño del datagrama no incluye la cabecera
- El campo TTL se sustituye por Hop Limit
- No hay campo Checksum, ya que se realiza por los protocolos superiores
- El campo opciones se realiza como cabeceras de extensión
- Los campos de fragmentación se eliminan de la cabecera y se implementan en cabeceras de extensión
- El campo Protocol se sustituye por Next Header

## ICMPv6

## ICMPv6: Introducción

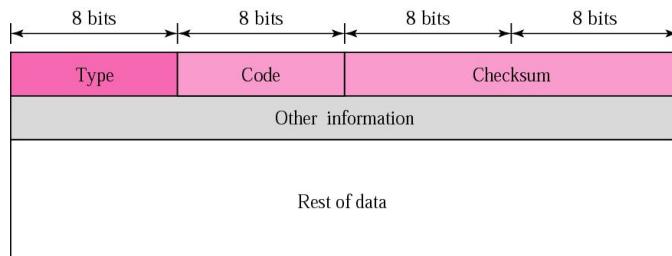
- ICMPv6 (RFC 4443) asume el papel de varios protocolos auxiliares en IPv4



Network layer in version 4

Network layer in version 6

- Protocolo orientado a mensajes
  - Mensajes de error
  - Mensajes de información, incluyendo el protocolo de descubrimiento de vecinos (RFC 4861) y la gestión de grupos multicast (RFC 3810)
- Los mensajes ICMPv6 tienen un formato común:



## ICMPv6: Mensajes de Error

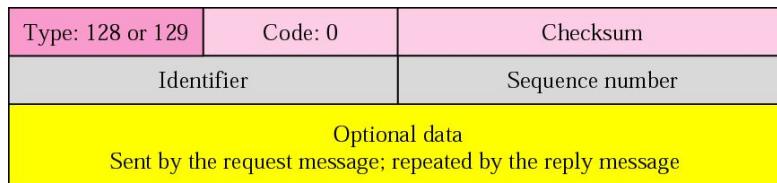
- Incluye errores relativos a (tipos de 0 a 127):
  - Destino inalcanzable (1)
  - Datagrama demasiado grande (2) → Path MTU Discovery, indica la MTU
  - Tiempo excedido (3)
  - Problema de parámetros (4)
- Ejemplo: Destino inalcanzable (1)
  - Si un encaminador o nodo no puede encaminar o entregar un datagrama (salvo porque haya congestión), se descarta y se envía este mensaje ICMP

Type: 1	Code: 0 to 4	Checksum
Unused (All 0s)		
Part of the received IP datagram including IP header plus the first 8 bytes of datagram data		

- Códigos: Sin ruta al destino (0), Comunicación no permitida (1), Fuera del ámbito del origen (2), Dirección inalcanzable (3), Puerto inalcanzable (4), Fallo de la dirección origen (5), Ruta rechazada (6)

# ICMPv6: Mensajes de Información

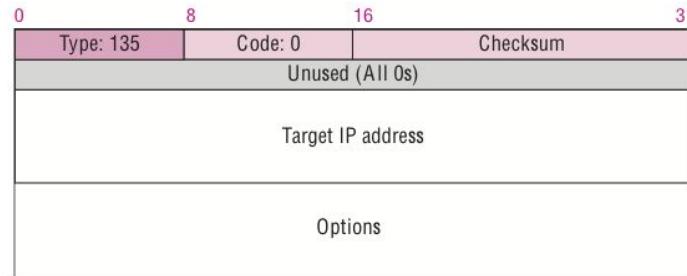
- Proporcionan información de diagnóstico (tipos de 128 a 255):
  - Echo request (128)
  - Echo reply (129)
  - Descubrimiento de vecinos
  - ...
- Ejemplo: Echo request/reply



- Identificador y Secuencia (16 bits cada uno): Sirven para identificar las respuestas, dependen de la implementación de ping
- Datos: deben copiarse en la respuesta

# ICMPv6: Solicitud de Vecino

- Este mensaje se genera para:
  - Averiguar la dirección física asociada a una dirección IP (como una petición ARP en IPv4), usando la dirección multicast de nodo solicitado (FF02::0:0:0:1:FF00::/104) como destino
  - Determinar si un nodo vecino sigue siendo alcanzable, usando la dirección unicast del interfaz como destino
  - Detectar si la dirección IP está duplicada, en el proceso de autoconfiguración
- Formato:



- Opciones: Dirección de enlace origen

# ICMPv6: Descubrimiento de Vecinos

- Protocolo multifunción que permite realizar operaciones de configuración
- Opera sobre nodos y encaminadores en el mismo enlace

## Descubrimiento de vecinos

- Resolución de direcciones (equivalente a ARP en IPv4) (IP nodo solicitado)
- Detección de direcciones duplicadas (IP nodo solicitado)
- Detección de vecino inalcanzable (IP unicast)
- Anuncio de cambios en la dirección de enlace (IP todos los nodos del enlace)
- Mensajes ICMPv6 Neighbor Solicitation (135) y Neighbor Advertisement (136)

## Descubrimiento de encaminadores

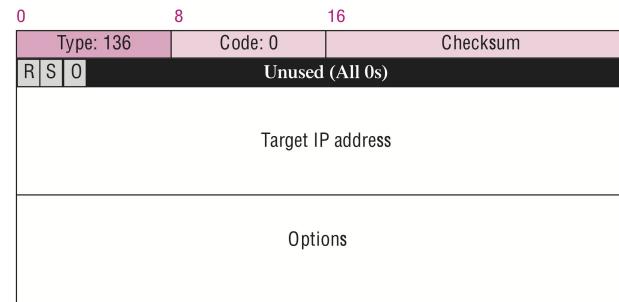
- Descubrimiento de encaminadores, prefijos y otra información de configuración de la red
- Mensajes ICMPv6 Router Solicitation (133) y Router Advertisement (134)

## Redirección

- Notificar a un nodo una ruta más adecuada para alcanzar un determinado destino
- Mensaje ICMPv6 Redirect (137)

# ICMPv6: Anuncio de Vecino

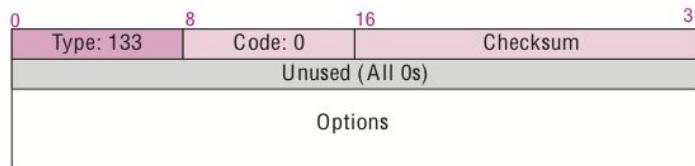
- Este mensaje se genera para:
  - Responder a un mensaje de solicitud de vecino (como una respuesta ARP en IPv4), con la dirección unicast del destinatario como destino
  - Anunciar un cambio en la dirección física de un interfaz, con la dirección multicast FF02::1 (todos los nodos del enlace local) como destino
- Formato:



- Flags: R (Router) indica si el emisor es un encaminador, S (Solicited) indica si el anuncio se envió como respuesta a una solicitud y O (Override) indica si se debe reemplazar la entrada existente en la cache (direcciones enlace local)
- Opciones: Dirección de enlace del interfaz (target)

# ICMPv6: Solicitud de Encaminador

- Este mensaje se genera tras la activación de un interfaz para:
  - Detectar los encaminadores y realizar la autoconfiguración del interfaz, con la dirección multicast FF02::2 (todos los encaminadores del enlace local) como destino
- Formato:



- Opciones: Dirección de enlace origen

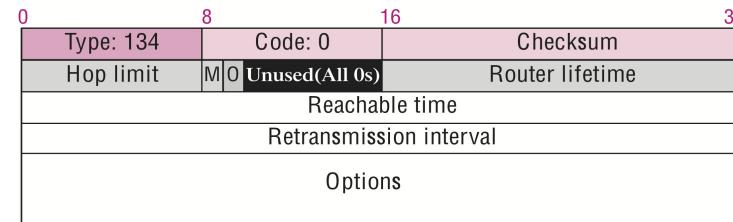
# ICMPv6: Autoconfiguración

- StateLess Address AutoConfiguration* (SLAAC)
- La autoconfiguración de un interfaz incluye:
  - El identificador de interfaz, generado según EUI-64 modificado o con las extensiones de privacidad
  - El prefijo anunciado por el encaminador
- Las opciones pueden incluir además información de DNS
- DHCPv6: Protocolo DHCP para IPv6

# ICMPv6: Anuncio de Encaminador

- Los envían los encaminadores para anunciar su presencia en la red:
  - Periódicamente, con la dirección multicast FF02::1 (todos los nodos del enlace local) como destino
  - Como respuesta a un mensaje de solicitud de encaminador, con la dirección multicast anterior o la dirección unicast del nodo solicitante como destino

- Formato:



- Flags: M (*Managed address configuration*) indica que se usa DHCPv6 para asignar direcciones y O (*Other configuration*), que se usa DHCPv6 para proporcionar otra información de configuración (ej. servidores DNS)
- Opciones: Dirección de enlace origen (interfaz del encaminador), MTU, prefijo de red, servidor DNS recursivo...

# Ejercicios: Preguntas Teóricas

¿Para qué se usa la dirección de red IPv6 ff02::1:ff61:db90?

- Para resolver la dirección física asociada a una de las IP de la máquina.
- Para comunicarse con las máquinas del sitio local.
- Para comunicarse con las máquinas del enlace.

¿Cómo se realiza la resolución de direcciones en IPv6?

- Mediante el uso del protocolo ARPV6.
- A partir del identificador extendido de 64 bits (EUI-64).
- Mediante el uso del protocolo ICMPv6.

Respecto a las opciones en el datagrama de IPv6, ¿cuál de las siguientes afirmaciones es cierta?

- Las opciones se usan solo cuando se realiza la fragmentación en origen.
- Las opciones se codifican como una cabecera adicional.
- No soporta opciones para acelerar el procesamiento de los routers.



## TEMA 1.5. Encaminamiento en Internet

### PROFESORES:

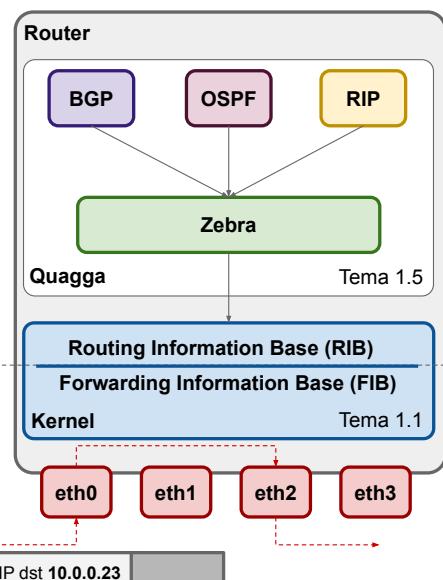
Rubén Santiago Montero  
 Eduardo Huedo Cuesta  
 Rafael Rodríguez Sánchez  
 Luis M. Costero

# El problema del encaminamiento

- En una red de conmutación de paquetes, el **encaminamiento** consiste en encontrar un camino, desde el origen al destino, a través de nodos de conmutación o encaminadores (*routers*) intermedios
- Caminos alternativos**
  - Es necesario decidir cuál es el mejor camino posible (*camino más corto*)
  - El *camino más corto* minimiza una métrica de encaminamiento
- Métricas de encaminamiento**
  - Número de saltos**: tiene en cuenta el número de encaminadores y/o redes intermedias que tiene que atravesar el paquete para alcanzar el destino
  - Distancia geográfica**: tiene en cuenta la distancia (en Km) que tiene que recorrer el paquete para alcanzar el destino
  - Retardo promedio**: tiene en cuenta el retardo de las líneas. Dado que éste es proporcional a la distancia, esta métrica es similar a la anterior
  - Ancho de banda**: tiene en cuenta la velocidad de transmisión de las líneas por las que tiene que circular el paquete
  - Nivel de tráfico**: tiene en cuenta el nivel de uso de las líneas, para intentar utilizar aquellas líneas con menor nivel de saturación
  - Combinación lineal de varias métricas

## Encaminamiento y Reenvío

- Plano de control**: Decide el mejor camino para los paquetes
- Plano de datos**: Reenvía los paquetes por el interfaz adecuado



```
# vtysh -c "show ip rip"
Codes: R - RIP, C - connected, S - Static, O - OSPF, B - BGP
...
Network      Next Hop     Metric  From           Tag  Time
C(i) 172.16.0.0/16  0.0.0.0   1       self          0
R(n) 172.17.0.0/16  172.16.0.2  2       172.16.0.2  0  02:31
R(n) 172.18.0.0/16  172.19.0.4  2       172.19.0.4  0  02:43
C(i) 172.19.0.0/16  0.0.0.0   1       self          0
```

```
# vtysh -c "show ip route"
Codes: K - kernel route, C - connected, S - static, R - RIP,
      O - OSPF, I - IS-IS, B - BGP, A - Babel,
      > - selected route, * - FIB route
C>* 172.16.0.0/16 is directly connected, eth0
R>* 172.17.0.0/16 [120/2] via 172.16.0.2, eth0, 00:00:29
R>* 172.18.0.0/16 [120/2] via 172.19.0.4, eth1, 00:00:17
C>* 172.19.0.0/16 is directly connected, eth1
```

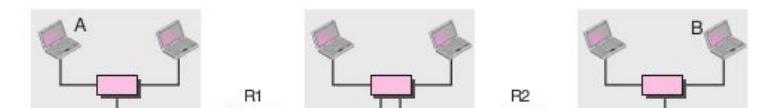
### Plano de control

### Plano de datos (o de forwarding)

```
# ip route
172.16.0.0/16 dev eth0 proto kernel scope link src 172.16.0.1
172.17.0.0/16 via 172.16.0.2 dev eth0 proto zebra metric 2
172.18.0.0/16 via 172.19.0.4 dev eth1 proto zebra metric 2
172.19.0.0/16 dev eth1 proto kernel scope link src 172.19.0.1
```

## Encaminamiento por Siguiente Salto

- Se basa en el **principio de optimalidad** de Bellman: Si el camino más corto entre dos encaminadores A y B es a través de C, entonces el camino más corto de C a B es a través de la misma ruta
- Para encaminar un paquete a lo largo del camino más corto, sólo es necesario conocer la identidad del siguiente encaminador inmediato a lo largo del camino



Destination	Route	Destination	Route	Destination	Route
Host B	R1, R2, Host B	Host B	R2, Host B	Host B	Host B

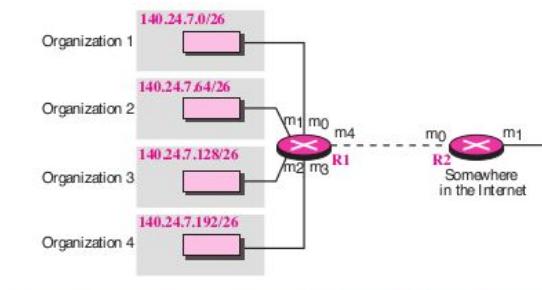
a. Routing tables based on route

Destination	Next Hop	Destination	Next Hop	Destination	Next Hop
Host B	R1	Host B	R2	Host B	---

b. Routing tables based on next hop

# Encaminamiento Escalable

- El encaminamiento escalable depende de controlar el tamaño de las tablas de rutas de los encaminadores
  - El encaminamiento con clase no es viable debido al gran número de redes (y, por tanto, entradas en las tablas) en Internet
- El encaminamiento en Internet se basa en:
  - CIDR, que permite agregación de direcciones y resumir las entradas
  - Encaminamiento jerárquico, que limita la información intercambiada



Mask	Network address	Next-hop address	Interface
/26	140.24.7.0	-----	m0
/26	140.24.7.64	-----	m1
/26	140.24.7.128	-----	m2
/26	140.24.7.192	-----	m3
/0	0.0.0.0	default router	m4

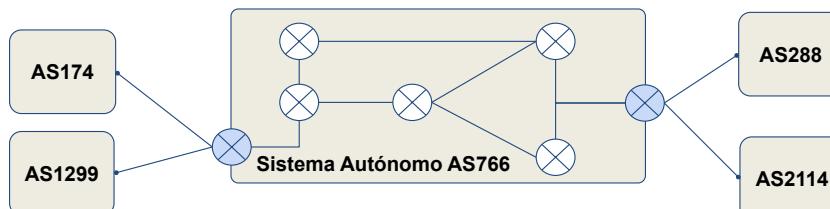
Routing table for R1

Mask	Network address	Next-hop address	Interface
/24	140.24.7.0	0.0.0.0	m0

Routing table for R2

# Encaminamiento en Internet

- Internet está organizado en **Sistemas Autónomos** (*Autonomous Systems*, AS)
  - Conjunto de redes y encaminadores gestionados por una misma autoridad
  - Se identifican mediante un número de AS (*AS Number*, ASN)
  - Hay más de 54.000 AS
- Los **encaminadores internos** del AS interconectan redes dentro del propio AS
  - Sólo conocen en detalle la organización del AS local
  - Protocolos IGP (*Interior Gateway Protocol*), ej. RIP y OSPF
- Los **encaminadores externos** o frontera (*border router*) del AS se conectan a otros AS
  - Conocen la ruta a otros AS
  - Protocolos EGP (*Exterior Gateway Protocol*), ej. BGP



# Tipos de Encaminamiento

## Encaminamiento estático

- Las tablas de encaminamiento se construyen manualmente considerando la topología de la red
- No se adapta a los cambios de la red

## Encaminamiento dinámico

- Las tablas de encaminamiento se construyen de forma automática, mediante el intercambio periódico de información entre los encaminadores
- Se adapta a los cambios de la red
- Las técnicas más comunes son:
  - Encaminamiento por vector de distancias (ej. RIP)
  - Encaminamiento por estado de los enlaces (ej. OSPF)



## AMPLIACIÓN DE SISTEMAS OPERATIVOS Y REDES

Grado en Ingeniería Informática / Doble Grado  
Universidad Complutense de Madrid

## Vector de Distancias: RIP (Routing Information Protocol)

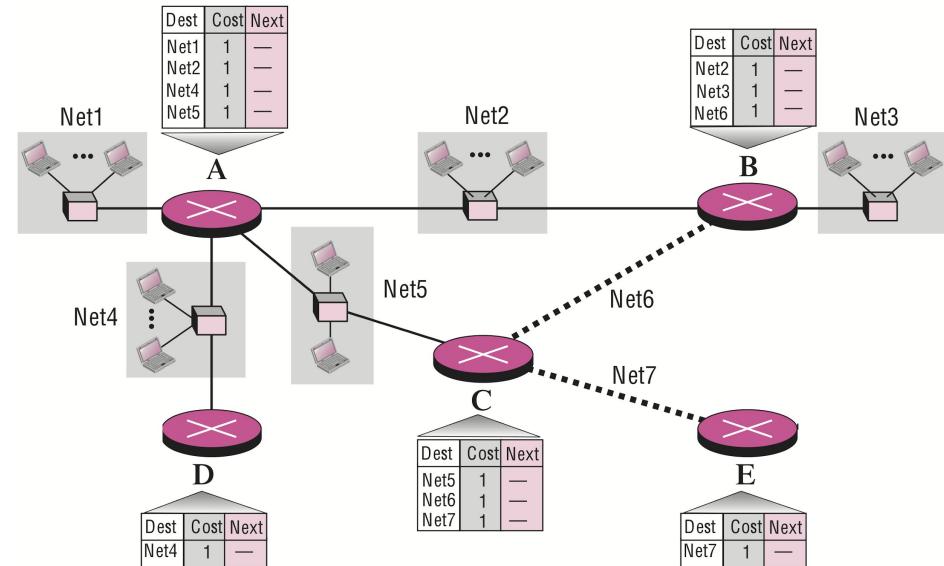
# Vector de Distancias

## Fundamentos

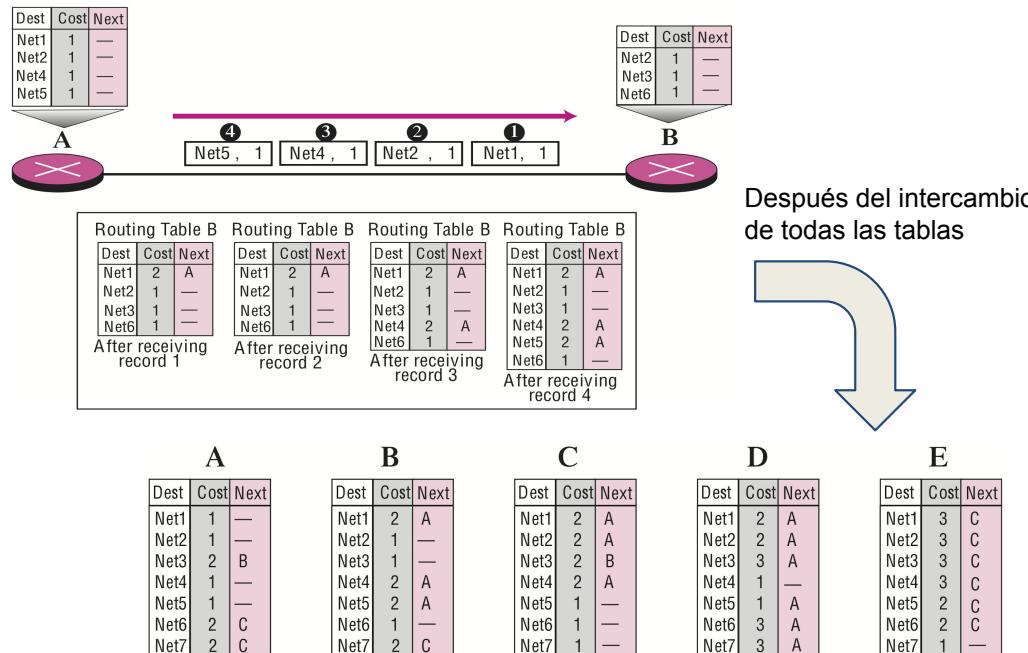
- Cada encaminador mantiene una tabla de encaminamiento con una entrada por cada posible destino en la red
- Cada entrada de la tabla contiene:
  - El destino (normalmente una red)
  - El siguiente salto (nodo o encaminador) para alcanzar dicho destino
  - La distancia o métrica para el destino, que suele ser el número de saltos
- Para construir la tabla de encaminamiento, los nodos intercambian periódicamente sus vectores de distancias (destinos y distancias) con sus vecinos
  - La distancia total al destino es la anunciada por el encaminador más la distancia al encaminador (normalmente, un salto)
  - Si la distancia total es menor que la actual, se sustituye la entrada
  - Si el siguiente salto de la entrada es el encaminador, se actualiza la entrada
    - La distancia total puede ser mayor debido a un cambio en la red
- El proceso iterativo de intercambio converge idealmente a los caminos óptimos
  - Este método también recibe el nombre de algoritmo de Bellman-Ford

# Vector de Distancias: Ejemplo

Inicialmente los encaminadores sólo conocen sus rutas directas



# Vector de Distancias: Ejemplo

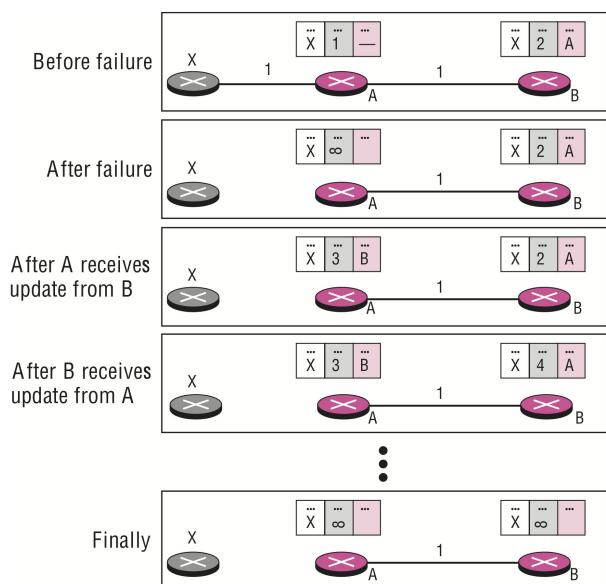


Después del intercambio de todas las tablas

# Vector de Distancias

## Problemas de convergencia. Cuenta a infinito

- Los cambios en la topología de la red deben propagarse a todos los encaminadores
- Cuando un enlace aumenta su distancia estos cambios tardan en propagarse
- Las actualizaciones para comunicar un enlace caído pueden no converger



# Vector de Distancias

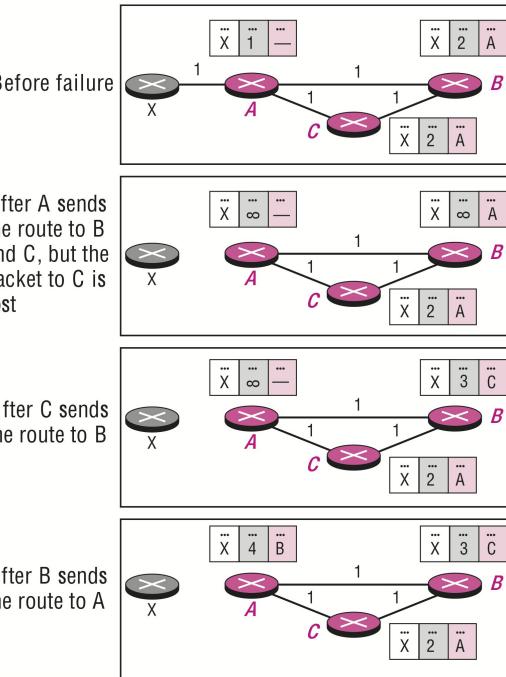
## Cuenta a infinito. Soluciones

- **Infinito pequeño (small infinity)**
  - Por ejemplo, en RIP el infinito se establece en 16 saltos (una distancia de 16 se considera inalcanzable y, por tanto, las rutas tienen un límite de 15 saltos)
- **Horizonte dividido (split horizon)**
  - Los destinos aprendidos a través de un determinado enlace nunca se difunden a través de dicho enlace
  - **Ejemplo:** El nodo B no enviará al nodo A información sobre el destino X
- **Horizonte dividido con ruta inversa envenenada (poisoned reverse)**
  - Los destinos aprendidos a través de un determinado enlace sí se difunden a través de dicho enlace, pero con distancia infinita
  - **Ejemplo:** El nodo B anunciará al nodo A que el destino X está a distancia infinita
- **Actualizaciones forzadas (triggered updates)**
  - Cuando un encaminador detecta una modificación en su tabla de rutas inmediatamente difunde esta información a sus vecinos
  - De esta forma, los cambios en la topología se propagan de forma rápida

# Vector de Distancias

## Problemas de convergencia. Bucles

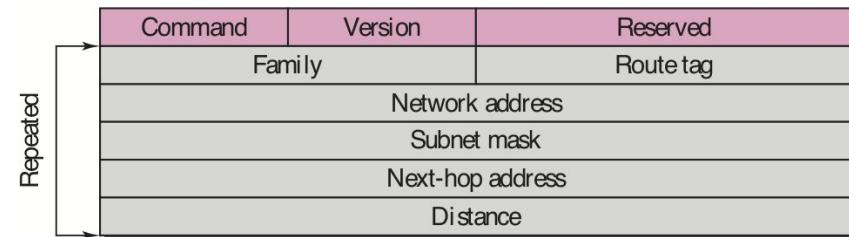
- En redes con bucles el algoritmo puede no converger
- Las técnicas de horizonte dividido no evitan el problema en este caso
- Las actualizaciones forzadas aceleran la convergencia



# Routing Information Protocol (RIP)

- Protocolo de encaminamiento interior (IGP) por vector de distancias
- Versiones y RFCs
  - RIP-1 → RFC 1058 (1993)
  - RIP-2 → RFC 2453 (1998)
  - RIPng (para IPv6) → RFC 2080 (1997)
- El vector de distancias incluye la siguiente información de encaminamiento:
  - La lista de destinos (redes) que son alcanzables por cada encaminador
  - La distancia a la que se encuentran dichos destinos, como número de saltos
- Los mensajes se encapsulan en datagramas UDP dirigidos al puerto 520
- El infinito se establece en 16 saltos y se pueden utilizar los siguientes mecanismos
  - Horizonte dividido
  - Horizonte dividido con ruta inversa envenenada
  - Actualizaciones forzadas
- La versión 2 añade:
  - Soporte para direcciones sin clase
  - Soporte para direccionamiento *multicast* (224.0.0.9)
  - Soporte para autenticación

# RIP: Formato del Mensaje



- **Command** (8 bits): Request (1) o Response (2)
- **Version** (8 bits): RIP-1 (1) o RIP-2 (2)
- **Family** (16 bits): TCP/IP (2) o entrada de autenticación (0xFFFF)
- **Route tag** (16 bits): Información adicional de ruta (ej. ASN, para separar rutas internas y externas) o algoritmo de autenticación, que puede ser una contraseña (2) o un resumen del mensaje con clave (3)
- **Network address** y **Subnet mask** (32 bits): Dirección de red sin clase
- **Next-hop address** (32 bits): Normalmente, es 0.0.0.0 para usar la dirección del remitente del mensaje, pero podría ser otro encaminador que no soporte RIP
- **Distance** (32 bits): Número de saltos al destino

# RIP: Mensajes

- **Mensajes de solicitud (REQUEST)**

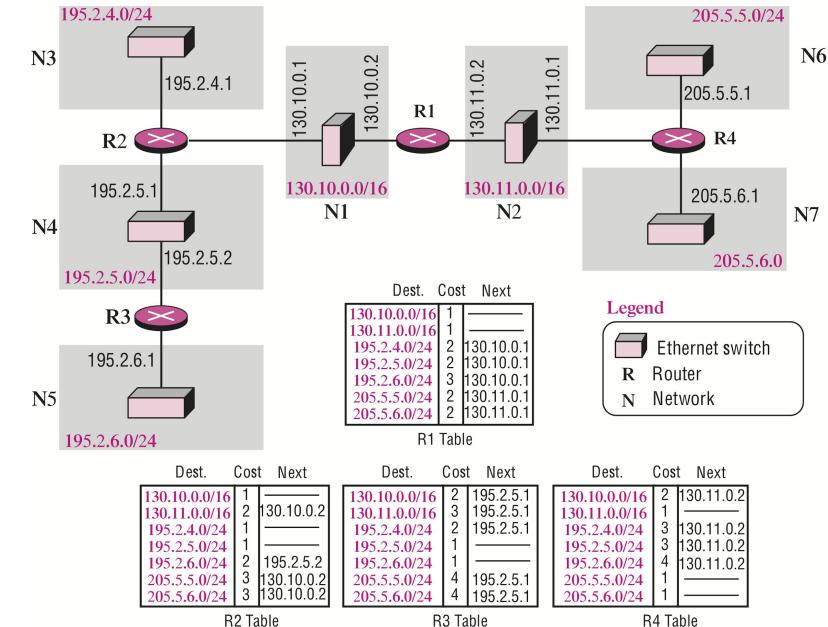
- Enviados cuando se conecta a la red → Network address = 0.0.0.0 (todas las entradas)
- Enviados cuando una entrada en la tabla expira → Network address = entrada expirada

- **Mensajes de respuesta (RESPONSE)**

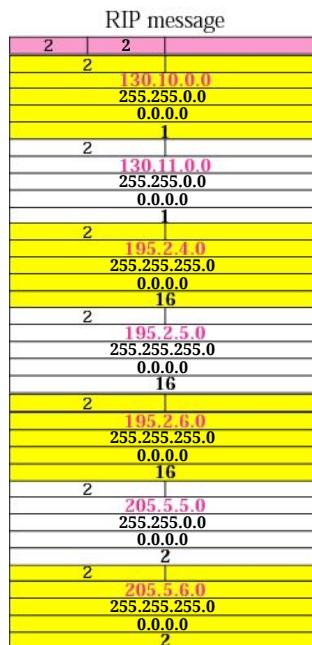
- Difundidos periódicamente (*broadcast* o *multicast*) con el vector de distancias
- Enviados en respuesta a una solicitud
- Enviados cuando la distancia a la red cambia (actualización forzada)

# RIP: Ejemplo

¿Qué mensaje RIP (RESPONSE) enviará R1 a R2?



# RIP: Ejemplo



# RIP: Temporizadores

- **Temporizador periódico (25-35 s)**

- Intervalo de envío de mensajes RESPONSE para anunciar el vector de distancias
- El protocolo RIP establece un valor de 30 s para este temporizador
  - En la práctica, se usa un valor aleatorio entre 25 y 35 s

- **Temporizador de expiración (180 s)**

- Controla el periodo de validez de una entrada de la tabla de encaminamiento
- Si no se recibe actualización de la entrada durante un intervalo de 180 s, la entrada deja de considerarse válida

- **Temporizador de “recolección de basura” (120 s)**

- Cuando una entrada de la tabla de rutas expira, el encaminador no la elimina inmediatamente de la tabla de encaminamiento
- La entrada se sigue anunciando con métrica 16 (destino inalcanzable) durante un periodo adicional de 120 s

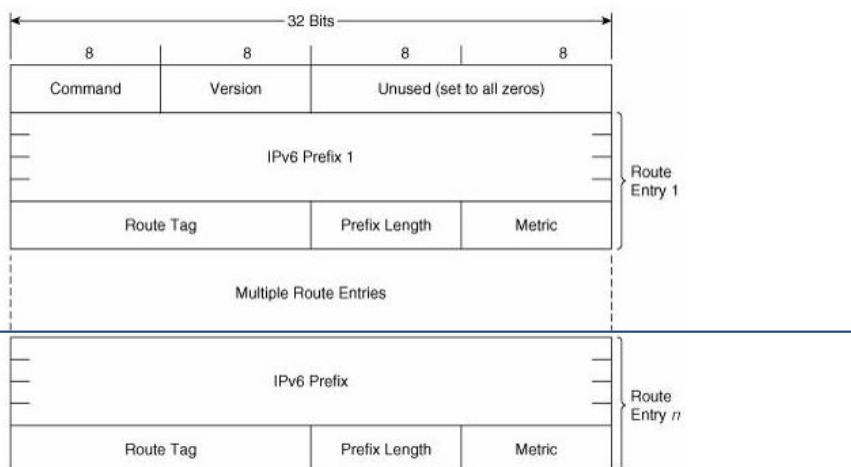
# RIP: Limitaciones

- Puede generar **gran cantidad de tráfico broadcast o multicast**, debido a la difusión periódica de los vectores de distancias (mensajes RESPONSE)
- **No admite métricas alternativas** al número de saltos
- Una vez calculadas las tablas, **no se admiten caminos alternativos** para equilibrar la carga de la red
- Cuando la red crece, los cambios pueden tardar bastante **tiempo en propagarse** hasta todos los puntos de la red
- El **infinito** se establece en 16 saltos
  - Redes grandes pueden necesitar más saltos

# RIPng: RIP para IPv6

- RIPng (RIP *next generation*) es la adaptación del protocolo RIP-2 para IPv6
- Diferencias con RIP-2:
  - Los mensajes RIPng se encapsulan en datagramas UDP dirigidos al puerto 521 y se difunden a la dirección IPv6 multicast FF02::9
  - El vector de distancias contenido en los mensajes de tipo RESPONSE, en lugar de direcciones de red IPv4, anuncia prefijos de red IPv6
  - La información de ruta contenida en un vector de distancias no incluye el campo Next Hop (casi duplicaría el tamaño de cada entrada)
    - En su lugar, se puede incluir una entrada especial (con 0xFF en el campo Metric) que afecta a las entradas siguientes
  - No implementa autenticación, sino que utiliza los mecanismos de cifrado y autenticación disponibles en IPv6

## RIPng: Formato del Mensaje



Entrada de la tabla de rutas:

- **IPv6 Prefix** (128 bits): prefijo de red IPv6 de la red destino anunciada
- **Prefix Length** (8 bits): longitud del prefijo de red anunciado
- **Route Tag** (16 bits) y **Metric** (8 bits): igual que en RIP-2



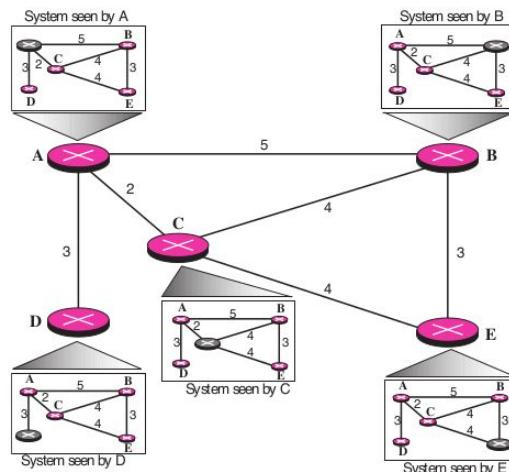
**AMPLIACIÓN DE SISTEMAS OPERATIVOS Y REDES**  
Grado en Ingeniería Informática / Doble Grado  
Universidad Complutense de Madrid

## Estado de los Enlaces: OSPF (Open Shortest Path First)

# Estado de Enlaces

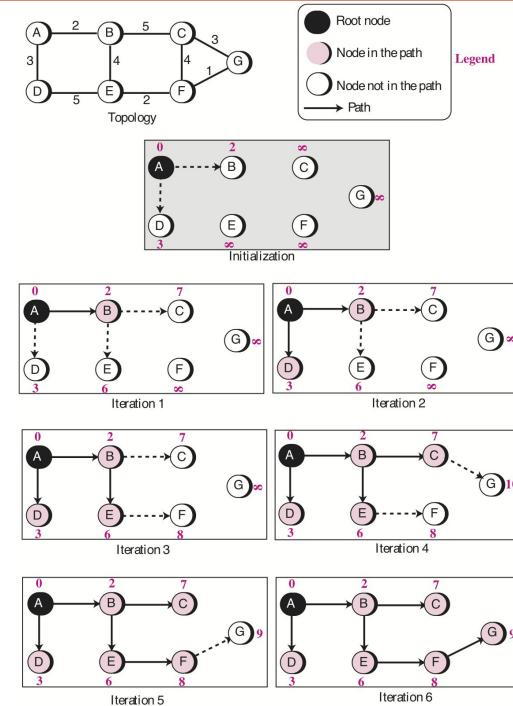
## Fundamentos

- Cada encaminador mantiene una base de datos (*link state database*) con la información sobre la topología exacta de la red
- Para construir esta base de datos, cada encaminador:
  - Identifica sus nodos vecinos y su distancia (estado de enlace)
  - Distribuye esta información a **todos** los nodos de la red (inundación)
- Usando la información completa de la red (grafo), cada nodo construye un mapa de rutas (árbol) con él mismo como origen (raíz) usando el algoritmo de Dijkstra



# Estado de Enlaces: Ejemplo

Calcular las rutas desde el nodo A y derivar la tabla de encaminamiento



# Open Shortest Path First (OSPF)

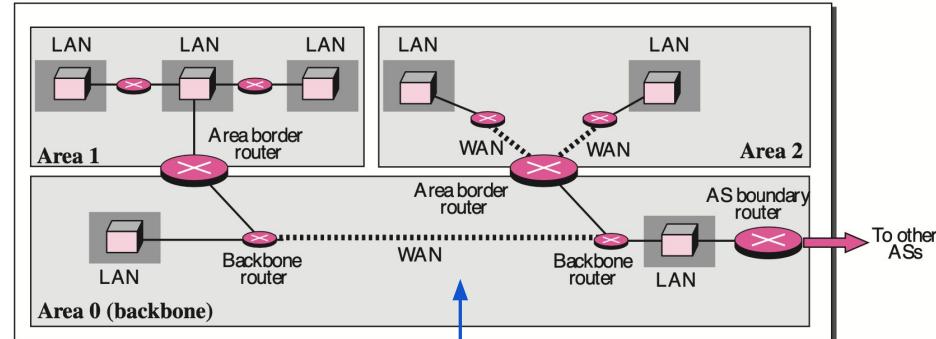
- Protocolo de encaminamiento interno (IGP) por estado de los enlaces
- Se desarrolló como alternativa a RIP para aliviar sus limitaciones:
  - Equilibrado de carga entre caminos equivalentes
  - Particionado lógico de la red para reducir la cantidad de información anunciada
  - Convergencia más rápida, por propagar inmediatamente los cambios en la red
  - Soporte para máscaras de longitud variable (VLSM) y CIDR
  - Soporte para autenticación de cualquier nodo que anuncie rutas
- Utiliza un protocolo propio de encapsulado (89) y direcciones *multicast*:
  - 224.0.0.5 o FF02::5 para todos los encaminadores OSPF de una red
  - 224.0.0.6 o FF02::6 para los encaminadores designados OSPF de una red
- Versiones y RFCs
  - OSPF version 2 → RFC 2328 (1998)
  - OSPF version 3 (para IPv6) → RFC 5340 (2008)

# OSPF: Áreas

Un **área** es una agrupación lógica de encaminadores y redes, con un identificador único (Area ID) de 32 bits

- Los encaminadores mantienen únicamente información de su área
- Limitan el número de intercambios de información de los enlaces

## Autonomous System (AS)



- El área 0 (*backbone*) existe en toda red OSPF
- La topología se crea conectando áreas adicionales al *backbone*, por lo que conecta todas las áreas

# OSPF: Encaminadores y Redes

## Encaminadores

- Cada encaminador tiene un identificador único (Router ID) de 32 bits en la red OSPF
- La información que se almacena y se intercambia depende del tipo:
  - **Internal Router (IR)**
    - Localizado en un área (todos sus interfaces están en el área)
    - Mantiene sólo información de la topología de su área
  - **Area Border Router (ABR)**
    - Conectado a dos o más áreas (una de ellas tiene que ser la 0)
    - Mantiene una DB para cada una de las áreas a las que está conectado
  - **Autonomous System Boundary Router (ASBR)**
    - Situado en la frontera del AS, transmite rutas externas a la red OSPF
    - Puede injectar rutas aprendidas mediante otro protocolo, como RIP

## Redes

- Establecen la frecuencia y el tipo de comunicaciones entre los encaminadores
- Entre otras, se definen redes punto-a-punto y multi-acceso

# OSPF: Vecindades y Adyacencias

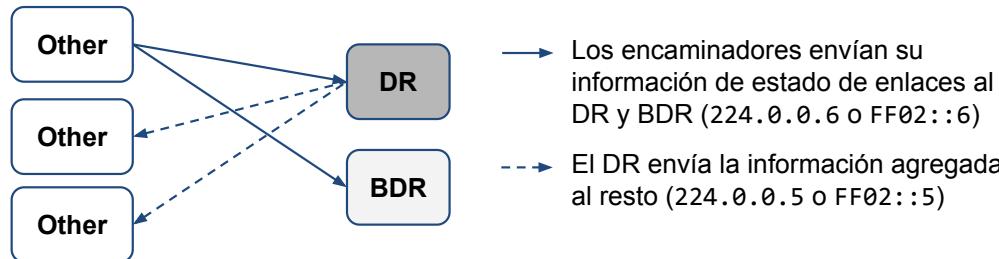
## Relación de vecindad

- Se establece entre encaminadores que comparten un enlace común, pertenecen a la misma área OSPF y usan el mismo mecanismo de autenticación
- Descubrimiento de vecinos mediante el protocolo OSPF Hello

## Relación de adyacencia

- Se establece entre encaminadores vecinos
  - Dos encaminadores adyacentes sincronizan su base de datos para establecer la adyacencia completa y posteriormente intercambian información de estado de los enlaces
  - Permite limitar la información intercambiada entre los encaminadores (no se comunican todos los vecinos)
- Se desarrolla según el tipo de red:
  - Punto-a-punto: entre los dos dispositivos vecinos
  - Multi-acceso: el encaminador designado (DR) y el encaminador designado de respaldo (BDR) son adyacentes al resto de los encaminadores de la red
- Selección de DR y BDR mediante el protocolo OSPF Hello

# OSPF: Vecindades y Adyacencias



- Los encaminadores envían su información de estado de enlaces al DR y BDR (224.0.0.6 o FF02::6)
- El DR envía la información agregada al resto (224.0.0.5 o FF02::5)

- El proceso de distribución de la información de estado de los enlaces es una optimización de la estrategia de inundación
- En caso de fallo del DR, el BDR asume sus funciones
- Los anuncios del DR (ej. después de una actualización) no son inmediatos, para solapar el envío de múltiples actualizaciones
- Los mensajes de actualización se confirman para asegurar la fiabilidad



## AMPLIACIÓN DE SISTEMAS OPERATIVOS Y REDES

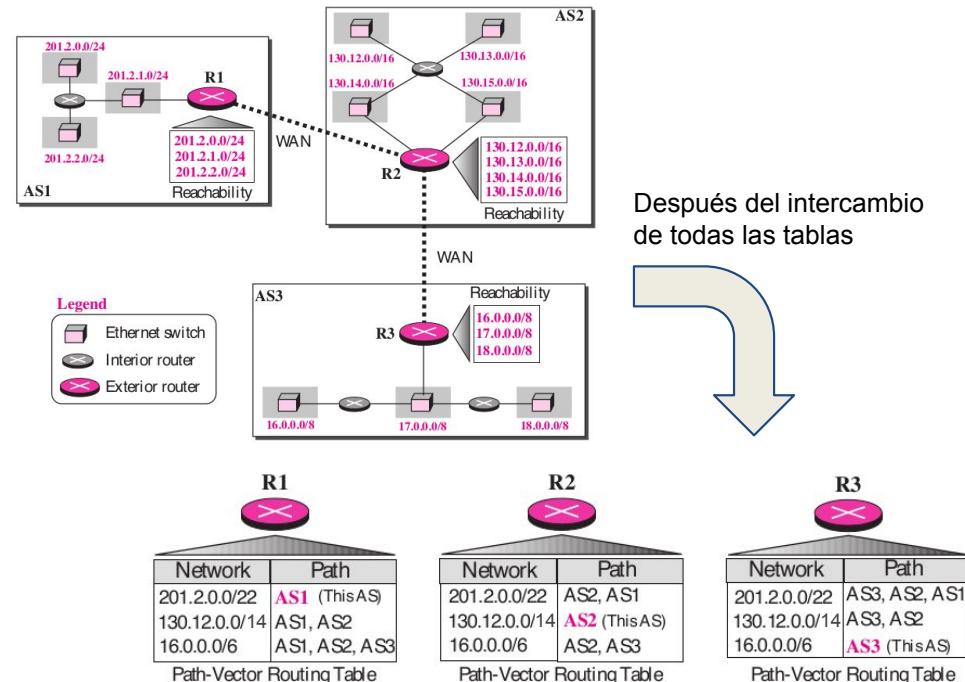
Grado en Ingeniería Informática / Doble Grado  
Universidad Complutense de Madrid

## Vector de Rutas: BGP (Border Gateway Protocol)

# Vector de Rutas

- Las técnicas anteriores no pueden aplicarse para el encaminamiento inter-AS
  - El encaminamiento por *vector de distancias* presenta inestabilidades con pocos saltos entre las redes y problemas de convergencia
  - El encaminamiento por *estado de enlaces* converge rápidamente pero requiere el intercambio de gran cantidad de información
- El encaminamiento por vector de rutas (Path Vector Routing) se basa en el encaminamiento por vector de distancias, e intenta resolver los problemas de convergencia para el encaminamiento inter-AS
  - A partir de la información sobre los destinos alcanzables en el AS, mediante un proceso de intercambio, cada encaminador obtiene:
    - La lista de destinos (redes) alcanzables
    - La *ruta* completa al destino, como lista de AS que han de atravesarse
  - Detección de bucles sencilla, descartando las rutas en las que el propio AS ya es parte del camino
  - Permite implementar políticas comprobando si un determinado AS es parte de la ruta
  - Uso de CIDR para agregar direcciones de red en las tablas de rutas

# Vector de Rutas: Ejemplo



# Border Gateway Protocol (BGP)

- Protocolo de encaminamiento exterior (Inter-AS) por vector de rutas
- La función principal de un sistema BGP es intercambiar información sobre las redes alcanzables (NLRI, *Network Layer Reachability Information*) con otros sistemas BGP
  - La información incluye la lista de AS atravesados por la propia información
  - Esta información es suficiente para construir un grafo de conectividad de AS para las redes alcanzables, libre de bucles
  - Cada AS puede aplicar ciertas políticas para aceptar y anunciar rutas en función de esa información (las políticas no forman parte de BGP)
- La comunicación entre encaminadores se realiza mediante TCP, puerto 179
- La versión actual, BGP-4, soporta CIDR y agregación de rutas

# BGP: Funcionamiento

- Los encaminadores intercambian sus tablas de rutas cuando establecen la conexión inicial y envían actualizaciones incrementales si las tablas cambian
- Mensajes:
  - OPEN:** Establecimiento de la sesión BGP (semipermanente)
    - Identificador de AS y de encaminador
    - Parámetros de configuración (tiempo *hold* y autenticación)
  - UPDATE:** Actualización incremental de la información de encaminamiento
    - Cada mensaje puede incluir una red alcanzable en CIDR con sus atributos, incluida la ruta, y una lista de redes retiradas (*withdrawn*)
  - NOTIFICATION:** Se envía a los vecinos cuando se detecta un error
    - Se cierra la sesión y se invalidan las rutas asociadas
    - Ejemplos: tiempo *hold* excedido, error en los mensajes, falta de atributos...
  - KEEPALIVE:** Para asegurar que la sesión permanezca activa
    - En respuesta a un mensaje OPEN y periódicamente para informar de la presencia del encaminador (no usa *keepalive* de TCP)
    - Si pasado un tiempo (*hold*) no se recibe información, se cierra la sesión



- Los mensajes UPDATE incluyen las redes alcanzables y atributos de cada ruta
  - Los atributos permiten evaluar caminos alternativos al mismo destino
  - Son generados por cada encaminador, que puede modificar los recibidos
- Tipos de atributos:
  - Bien conocidos (well-known)**: Deben ser admitidos por todas las implementaciones BGP
    - Pueden ser obligatorios (*mandatory*) o discrecionales (*discretionary*)
    - Los atributos obligatorios se deben incluir en cada actualización
  - Opcionales**: Son específicos de cada implementación
    - Pueden ser transitivos (*transitive*) o no
    - Los atributos transitivos se debe incluir en las actualizaciones aunque no sean implementados por el encaminador
- Ejemplos de atributos bien conocidos y obligatorios (*well-known mandatory*):
  - ORIGIN**: Origen de la información de ruta (IGP, EGP o INCOMPLETE)
    - No debe modificarse por otro encaminador BGP
  - AS\_PATH**: La ruta como secuencia de AS
  - NEXT\_HOP**: Dirección IP del siguiente salto para alcanzar el destino

## Arquitectura de Internet

## Interconexión de Sistemas Autónomos

**Tier 1:** Tienen acceso a cualquier red sin pagar. Backbone de Internet. Infraestructura global de gran tamaño. Ej. [Telxius](#), [Cogent](#), [cables sub](#).

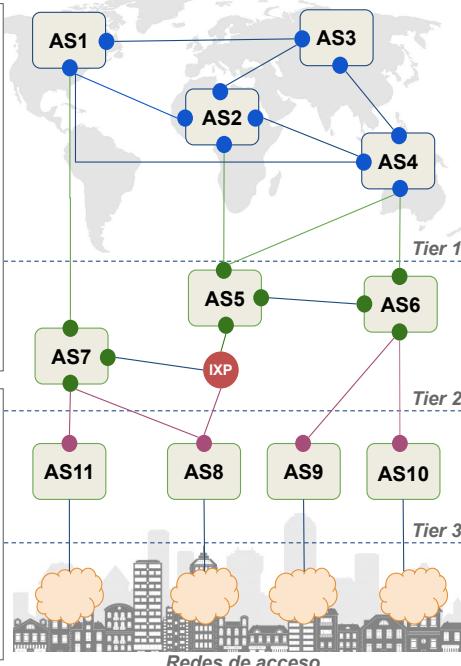
**Tier 2:** Proveedores (*carriers*) de tamaño medio-grande que pagan para llevar su tráfico a través de algún proveedor. Normalmente a un salto del *backbone* y con puntos de presencia en un solo continente. Ej. [Aire Networks](#)

**Tier 3:** Proveedores de acceso a Internet (ISP) que pagan por el tráfico que generan. Infraestructura a nivel regional o nacional.

**Stub AS:** Conectado únicamente a otro AS, es destino u origen del tráfico (ej. AS9)

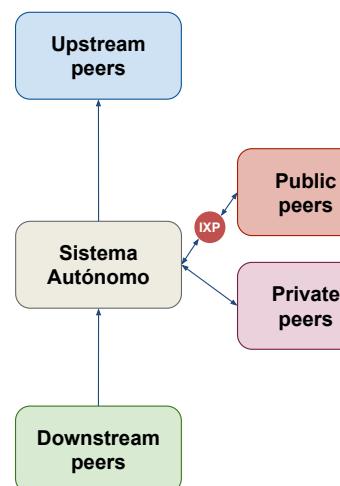
**Transit AS:** Conectado a varios AS, permite tráfico de tránsito, es decir, entre otros dos AS (ej. AS de tier 1 y 2)

**Multihomed AS:** Conectado a varios AS por redundancia y equilibrado, es destino u origen del tráfico, sin permitir tráfico de tránsito (ej. AS8)



## Peering entre Sistemas Autónomos

El **peering** es la relación que establecen dos AS por la que intercambian información de encaminamiento



### Upstream Peering (proveedores)

- El AS consume servicios de tránsito a Internet
- Los AS *upstream* envían la información de rutas de las redes a las que tienen acceso
- Aceptan las rutas del AS y sus clientes

### Public/Private Peering (entre iguales)

- Público. En puntos neutros (*Internet eXchange Points*, IXP). Ej. [Espanix](#).
- Privado. Mediante enlaces directos dedicados
- Intercambian prefijos de sus redes y de sus clientes
- Estos prefijos no se redistribuyen *upstream*
- Suele ser relaciones gratuitas

### Downstream Peering (clientes)

- El AS proporciona servicios de tránsito a Internet
- Debe distribuir los prefijos de sus clientes a sus peers

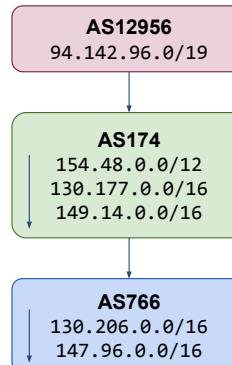
# Peering: Ejemplo

# Ejercicios: Preguntas Teóricas

Usando la herramienta [bgpvview.io](#):

- Determinar el AS al que pertenece la red de la UCM (147.96.0.0/16)
- ¿Cuántos proveedores *upstream* tiene el AS?
- ¿Qué clientes *downstream* tiene el AS?
- Usando también la [herramienta looking-glass](#) de Telxius, determinar los AS y redes atravesados para alcanzar www.ucm.es (147.96.1.15) desde Buenos Aires

```
1 94.142.116.117 28 msec 29 msec 26 msec
2 be9-grtmianal.net.telefonicaglobalsolutions.com (94.142.119.188) 122 msec
3 be3400.ccr21.mia01.atlas.cogentco.com (154.54.47.17) 122 msec
6 be3482.ccr41.at101.atlas.cogentco.com (154.54.24.145) 146 msec
7 be2112.ccr41.dca01.atlas.cogentco.com (154.54.7.157) 160 msec
8 be2331.ccr31.bio02.atlas.cogentco.com (154.54.85.242) 228 msec
9 be2324.ccr31.mad05.atlas.cogentco.com (154.54.61.130) 230 msec
10 be2475.rcr51.b015537-1.mad05.atlas.cogentco.com (130.117.0.218) 230 msec
11 149.14.242.226 246 msec
12 redimadrid-cieamt-router.rediris.es (130.206.212.106) 233 msec
13 redimadrid-cieamt-router.rediris.es (130.206.212.106) 233 msec
```



En el protocolo RIP con horizonte dividido, los anuncios del vector de distancias enviados por un enlace incluyen...

- Las redes alcanzables y su distancia.
- Las redes alcanzables y su distancia si ésta es menor que 16.
- Las redes alcanzables no aprendidas por ese enlace y su distancia.

¿Cuál de las siguientes afirmaciones sobre el protocolo BGP es cierta?

- Los encaminadores construyen un grafo de AS completo de la red.
- Los encaminadores intercambian la lista de AS a una red destino.
- Los encaminadores intercambian los AS alcanzables y el número de saltos para llegar a ellos.

En una red de encaminadores que usan el protocolo OSPF, el área 0 es...

- La que interconecta todas las demás áreas de la topología del AS.
- En la que están los encaminadores de frontera de AS (ASBR).
- El área del AS en la que están los clientes de red.

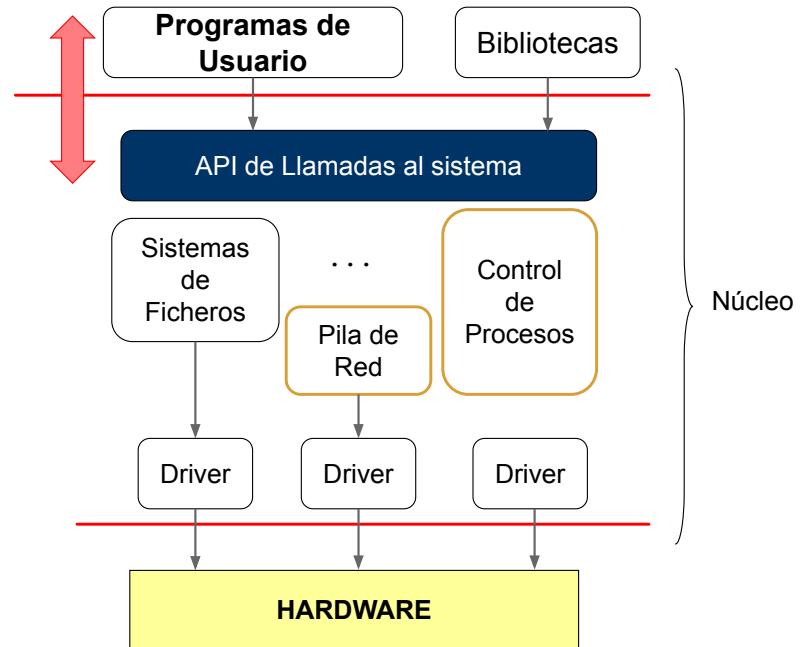


## TEMA 2.1. Introducción a la Programación de Sistemas

### PROFESORES:

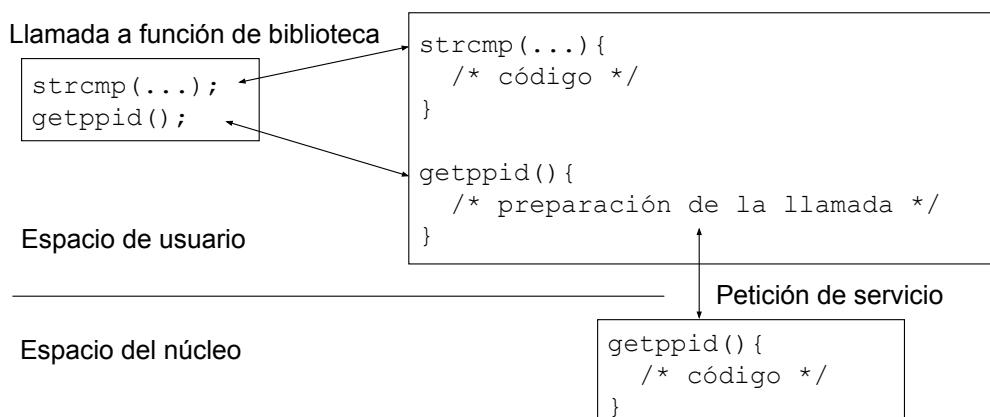
Rubén Santiago Montero  
Eduardo Huedo Cuesta  
Rafael Rodríguez Sánchez

## Introducción: Arquitectura del sistema



## Llamadas al Sistema y Funciones de Biblioteca

- Desde el punto de vista del programador no hay diferencia. Sin embargo:
  - Una llamada al sistema es una función de la biblioteca estándar de C que solicita un servicio del sistema (*trap*), que se resuelve en el núcleo del SO
  - Una función de biblioteca no interacciona de forma directa con el sistema (debe usar llamadas al sistema)



## Llamadas al Sistema y Funciones de Biblioteca

- Las llamadas al sistema y las funciones de biblioteca están documentadas en las páginas de manual (ver `man man`):
  - Sección 1: Comandos y aplicaciones
  - Sección 2: Llamadas al sistema**
  - Sección 3: Funciones de biblioteca**
  - Sección 4: Dispositivos y ficheros especiales
  - Sección 5: Formatos de ficheros y convenciones
  - Sección 6: Demostraciones y juegos
  - Sección 7: Miscelánea (convenciones y protocolos, juegos de caracteres, jerarquía del sistema de ficheros...)
  - Sección 8: Comandos de administración del sistema (superusuario)
  - Sección 9: Documentación del núcleo y desarrollo de drivers
- El formato general de consulta es: `man [section] comando`
- La sección del manual se especifica seguida del comando, en la forma: `open(2)`
- El uso de `-k keyword` es útil para buscar páginas específicas
- Puede ser necesario consultar los ficheros en `/usr/include/`

# API del Sistema

- *Application Programming Interface (API)*: Conjunto de funciones y rutinas agrupadas con un propósito común
- Consideraciones generales en el uso de un API:
  - ¿Qué fichero de cabecera necesita (`#include`)?
  - ¿Qué tipo de datos devuelve la función?
  - ¿Cuáles son los argumentos de la función?
    - Tipos de datos
    - Paso por valor o por referencia
  - ¿Qué significado tiene el valor de retorno de la función?
  - ¿Qué significado tienen los argumentos de la función?
  - ¿Cómo se tiene que gestionar la memoria de las variables?
  - ¿Cómo de portable es la función?
  - ¿Cómo puede fallar la función?

# Llamadas al Sistema y de Biblioteca

Llamadas al Sistema	Función de Biblioteca
Sección de manual	2
Área de ejecución	Usuario/Núcleo
Espacio de parámetros	No se reserva
Código de error	-1 + <code>errno</code>
	NULL + no <code>errno</code>

## API del Sistema: Traza de Llamadas

- Trazar las llamadas al sistema realizadas por un programa:

```
strace [opciones] comando [argumentos]
```

  - Ejecuta el comando hasta que termina, interceptando las llamadas al sistema que realiza y las señales que recibe
  - Permite analizar el comportamiento de programas de los que no se dispone el código fuente
  - En cada línea se muestra la llamada al sistema realizada, los argumentos de la llamada y el valor de retorno
  - Opciones:
    - -c: Recopila el tiempo, las llamadas y errores producidos mostrando un resumen
    - -f: Traza los procesos hijos a medida que se crean
    - -T: Muestra el tiempo de cada llamada
    - -e trace=call: Selección del tipo de llamadas al sistema trazadas (process, network, IPC, signal o file)
    - -e write=fd: realiza un volcado completo de los datos escritos en el descriptor de ficheros

## Gestión de Errores

- Imprimir un mensaje de error:

```
void perror(const char *s);
```

  - Imprime por la salida de error (`stderr`) un mensaje que describe el último error encontrado en una llamada al sistema o función de biblioteca
  - El formato de salida es:

s	:	Mensaje de error	\n
---	---	------------------	----
  - En la cadena debe incluirse el nombre de la función que produjo el error
- El código de error se obtiene de la variable `errno`, que se fija cuando se produce un error, pero no se borra cuando la llamada tiene éxito:

```
int errno;
```

  - Por convenio, las llamadas al sistema devuelven -1 cuando se produce un error
    - Muchas funciones de biblioteca también lo hacen
  - Devolver una cadena que describe el número de error:

```
char *strerror(int errnum)
```

```
<stdio.h>
<errno.h>
<string.h>
```

POSIX+ANSI-C

# Información del Sistema

- Obtener información sobre el sistema operativo:

```
int uname(struct utsname *buffer);

struct utsname {
    char sysname[]; /* Nombre del SO (ej. "Linux") */
    char nodename[]; /* Nombre del host */
    char release[]; /* Release del SO (ej. "3.10.0") */
    char version[]; /* Versión del SO (fecha) */
    char machine[]; /* Hardware (ej. "x86_64") */

#ifndef _GNU_SOURCE
    char domainname[]; /* Nombre de dominio (con -D) */
#endif
}
```

<sys/utsname.h>
SV+POSIX

- Devuelve la información en la estructura apuntada por `buffer` (paso por referencia)
- Código de error: **EFAULT** (`buffer` no es válido)
- El comando `uname` proporciona acceso a esta funcionalidad
- Parte de la información está accesible vía `sysctl` y vía  
`/proc/sys/kernel/{ostype,hostname,osrelease,version,domainname}`

# Información del Sistema

- Obtener información de configuración del sistema de ficheros:

```
long pathconf(char *path, int name);
long fpathconf(int fd, int name);
```

<unistd.h>
POSIX

- El parámetro `name` puede ser:
  - `_PC_LINK_MAX`: Número máximo de enlaces
  - `_PC_NAME_MAX`: Longitud máxima del nombre de fichero
  - `_PC_PATH_MAX`: Longitud máxima de la ruta relativa
  - `_PC_CHOWN_RESTRICTED`: Devuelve un valor no nulo si el cambio del propietario del fichero está restringido
  - `_PC_PIPE_BUF`: Número máximo de bytes que pueden escribirse atómicamente en la tubería
- La función devuelve el límite asociado con el parámetro o -1 en caso de que no exista (no modifica `errno`) o en caso de error (sí establece `errno`)
- El comando `getconf` proporciona acceso a esta funcionalidad

# Información del Sistema

- Obtener información de configuración del sistema:

```
long sysconf(int name);
```

- El argumento `name` puede ser:
  - `_SC_ARG_MAX`: Longitud máxima de argumentos en funciones `exec()`
  - `_SC_CLK_TCK`: Número de ticks de reloj por segundo (Hz)
  - `_SC_OPEN_MAX`: Número máximo de ficheros abiertos por proceso
  - `_SC_PAGESIZE`: Tamaño de página en bytes
  - `_SC_CHILD_MAX`: Número máximo de procesos simultáneos por usuario
- Devuelve el valor del parámetro o -1 en caso de error, pero no establece el valor de `errno`

- El comando `getconf` proporciona acceso a esta funcionalidad

<unistd.h>
POSIX

# Información del Sistema

- Obtener información de configuración del sistema de ficheros:

```
long pathconf(char *path, int name);
long fpathconf(int fd, int name);
```

<unistd.h>
POSIX

- El parámetro `name` puede ser:
  - `_PC_LINK_MAX`: Número máximo de enlaces
  - `_PC_NAME_MAX`: Longitud máxima del nombre de fichero
  - `_PC_PATH_MAX`: Longitud máxima de la ruta relativa
  - `_PC_CHOWN_RESTRICTED`: Devuelve un valor no nulo si el cambio del propietario del fichero está restringido
  - `_PC_PIPE_BUF`: Número máximo de bytes que pueden escribirse atómicamente en la tubería
- La función devuelve el límite asociado con el parámetro o -1 en caso de que no exista (no modifica `errno`) o en caso de error (sí establece `errno`)
- El comando `getconf` proporciona acceso a esta funcionalidad

# Información del Usuario

- Obtener los identificadores de un proceso:

```
uid_t getuid(void);
gid_t getgid(void);
uid_t geteuid(void);
gid_t getegid(void);
```

<unistd.h>
<sys/types.h>
BSD+POSIX

- Los procesos disponen de un identificador de usuario (**UID**) y de grupo (**GID**), que corresponden a los identificadores del propietario del proceso que, en general, se heredan del proceso que lo creó
  - Estos identificadores se denominan **UID y GID reales**
- Además, los procesos disponen de un identificador de usuario **efectivo (EUID)** y de grupo **efectivo (EGID)**, que son los que se comprueban para conceder permisos
  - Generalmente, los identificadores reales y efectivos coinciden
  - Sin embargo, si el fichero de programa tiene los bits `setuid` o `setgid` activos, el EUID o el EGID del proceso creado se cambian al usuario o grupo del fichero

# Información del Usuario

- Obtener **información de un usuario** de la base de datos de contraseñas:

```
struct passwd *getpwnam(const char *name);
struct passwd *getpwuid(uid_t uid);

struct passwd {
    char *pw_name; /* Nombre de usuario */
    char *pw_passwd; /* Contraseña */
    uid_t pw_uid; /* Identificador de usuario */
    gid_t pw_gid; /* Identificador de grupo */
    char *pw_gecos; /* Descripción del usuario */
    char *pw_dir; /* Directorio "home" */
    char *pw_shell; /* Shell */
}
```

<pwd.h>
<sys/types.h>
SV+POSIX+BSD

- Devuelven un puntero a una estructura asignada estáticamente que puede sobreescribirse (hay versiones **reentrantes**)
- Devuelven NULL, si no se encuentra el usuario o si se produce algún error (**ENOMEM** si no puede reservar memoria para la estructura)
- Con **shadow passwords** es necesario utilizar las funciones `getspnam`

# Información de la Hora del Sistema

- Obtener el tiempo en segundos desde el *Epoch*:

```
time_t time(time_t *t);
```

- El *Epoch* se refiere a 1970-01-01 00:00:00 +0000, UTC
- Si t no es NULL, el resultado también se almacena en la variable apuntada

<time.h>
SV+BSD+POSIX

- Obtener y establecer la fecha del sistema:

```
int gettimeofday(struct timeval *tv,
                 struct timezone *tz);
int settimeofday(const struct timeval *tv,
                const struct timezone *tz);

struct timeval {
    long tv_sec; /* segundos relativos a Epoch */
    long tv_usec; /* microsegundos */
}
```

- `gettimeofday` devuelve la fecha en la estructura apuntada por `tv` (paso por referencia)
- La estructura `timezone` está obsoleta y `tz` debe ponerse a NULL, de forma que la estructura correspondiente ni se modifica ni se retorna
- Únicamente el superusuario puede modificar la fecha del sistema

<unistd.h>
<sys/time.h>
SV+BSD

# Información de la Hora del Sistema

- Obtener el tiempo desglosado UTC (*Coordinated Universal Time*) o en la zona horaria local:

```
struct tm *gmtime(const time_t *time);
struct tm *localtime(const time_t *time);

struct tm {
    int tm_sec; /* segundos 0-59 */
    int tm_min; /* minutos 0-59 */
    int tm_hour; /* horas 0-23 */
    int tm_mday; /* día del mes 1-31 */
    int tm_mon; /* mes 0-11 */
    int tm_year; /* años desde 1900 */
    int tm_wday; /* día de la semana (Dom.) 0-6 */
    int tm_yday; /* día del año (1-1) 0-365 */
    int tm_isdst; /* horario verano/inviero */
};
```

- Devuelven un puntero a una estructura asignada estáticamente que podría sobreescribirse (hay versiones reentrantes)

<time.h>
SV+BSD+POSIX

# Información de la Hora del Sistema

- Formatear fecha y hora:

```
size_t strftime(char *s, size_t max,
                const char *format, const struct tm *tm);
```

- El parámetro `format` es una cadena donde:

%a: Día de la semana abreviado (idioma sistema)

%A: Día de la semana completo

%b: Mes abreviado

%B: Mes completo

%d: Día del mes en decimal

%j: Día del año en decimal (24)

%H: Hora en decimal (24)

%I: Hora en decimal (12)

%M: Minutos es decimal

%S: Segundos en decimal

%n: Retorno de carro

%p: PM, AM

%r: Hora en a.m./p.m. (equivalente a "%I:%M:%S %p")

<time.h>
SV+BSD+POSIX

- Devuelve la longitud de la cadena generada o 0 si supera los `max` bytes



¿Qué efecto tiene que un ejecutable tenga activado el permiso SETUID?

- El proceso se creará con permisos de superusuario.
- El proceso se creará con usuario (UID) igual al propietario del fichero.
- El proceso se creará con usuario efectivo (EUID) igual al propietario del fichero.

Una función de biblioteca...

- no puede usar internamente llamadas al sistema.
- es una función de la biblioteca estándar de C que solicita un servicio del sistema.
- no interacciona de forma directa con el sistema.

## Material adicional

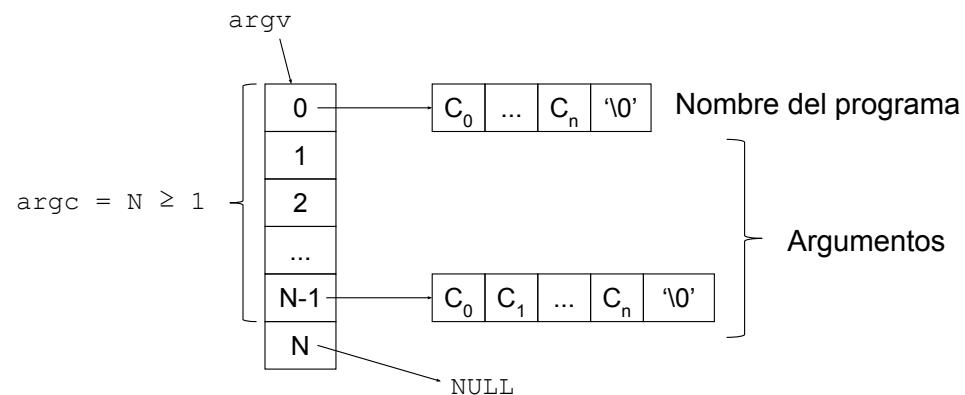
## Estándares de Programación

- **ANSI-C o ISO-C:** Estándar de programación adoptado por ANSI (*American National Standards Institute*) y posteriormente por ISO (*International Standardization Organization*). Es el estándar más general. La opción `-ansi` hace que el compilador lo cumpla de forma estricta.
- **BSD (Berkeley Software Distribution):** Desarrollado durante los 80 en la Universidad de California Berkeley. Sus contribuciones más importantes son los enlaces simbólicos, los sockets, la llamada `select...`
- **SVID (System V Interface Definition):** Descripción formal de las distribuciones comerciales de UNIX de la compañía AT&T, como System V Release 4 (SVr4). Su principal contribución son los mecanismos IPC.
- **POSIX (Portable Operating System Interface):** Estándares IEEE e ISO derivados de varias versiones de UNIX, principalmente de SVID. Incluye ANSI-C. Describe llamadas al sistema y de biblioteca, especifica la semántica detallada de la *shell* y un conjunto mínimo de comandos, así como interfaces detallados para varios lenguajes de programación.
- **GNU (GNU's Not Unix!):** Sistema operativo de tipo UNIX de software libre con licencia GNU GPL (*General Public License*). La combinación del software GNU y el núcleo de Linux es GNU/Linux.

## Argumentos del Programa

- Definición del **programa principal**:

```
int main(int argc, char **argv);  
int main(int argc, char *argv[]);
```



# Argumentos del Programa

# Argumentos del Programa

- POSIX recomienda las siguientes convenciones para los argumentos de línea de comandos:
  - Los argumentos se consideran opciones si empiezan con un guión (-)
  - Los nombres de las opciones son un único carácter alfanumérico
  - Se pueden indicar varias opciones tras un guión en un solo elemento si las opciones no toman argumentos. Por tanto, `-abc` es equivalente a `-a -b -c`
  - Ciertas opciones requieren un argumento, como `-o name`. El espacio entre la opción y el argumento es opcional. Por tanto, `-o foo` es equivalente a `-ofoo`
  - Normalmente, primero se indican las opciones y después el resto de argumentos
  - El argumento `--` termina las opciones. Los argumentos que le siguen se tratan como no opciones, incluso si empiezan por un guión
  - Un único guión se interpreta como un argumento ordinario. Por convención, se usa para especificar `stdin` o `stdout`
  - Las opciones se pueden proporcionar en cualquier orden o aparecer varias veces. La interpretación se deja al programa
- Las opciones largas (extensión de GNU) consisten en dos guiones seguidos de un nombre (que puede abreviarse) compuesto por caracteres alfanuméricos y guiones
  - Se puede especificar un argumento con `--name=value`

<unistd.h>
POSIX

- Procesar los argumentos de un programa:

```
extern char *optarg;
extern int optind, opterr, getopt;

int getopt(int argc, const char *argv [],  
          const char *options);
```

- options: Cadena que contiene las opciones válidas para el programa. Si al carácter le sigue `:', indica que esa opción usa un argumento
- optind: Índice que apunta al primer argumento que no es una opción
- opterr: Si el valor de esta variable no es nulo, `getopt()` imprime un mensaje de error cuando encuentre una opción desconocida
- getopt: Cuando se encuentra una opción desconocida o se detecta la falta de un argumento, la opción en cuestión se almacena en esta variable. Útil para mostrar mensajes propios de error
- optarg: Apunta al valor del argumento de la opción

# Argumentos del Programa

- Funcionamiento de `getopt(3)`:
  - Permuta los contenidos a medida que los trata de forma que los argumentos no-opciones se encuentran al final del array `argv`
  - Devuelve el siguiente carácter opción
  - Si no hay más devuelve -1. Para comprobar que no existen más argumentos no-opciones comparar `argc` con `optind`
  - Cuando la opción tiene un argumento, `getopt()` establece el puntero `optarg` (normalmente no es necesario copiarlo ya que es un puntero a `argv`, que no se modifica)
  - Cuando se encuentra una opción no válida o una opción que le falta argumento, devuelve el carácter `?' y establece `optopt` a la opción incorrecta
  - En caso de error si `opterr` no es cero se muestra un mensaje de error en la salida de error estándar



## TEMA 2.2. Sistema de Ficheros

### PROFESORES:

Rubén Santiago Montero  
Eduardo Huedo Cuesta  
Rafael Rodríguez Sánchez

# Características de los Sistemas de Ficheros

### Desde el punto de vista del usuario

- Colección de ficheros y directorios usados para guardar y organizar la información

### Desde el punto de vista del sistema operativo

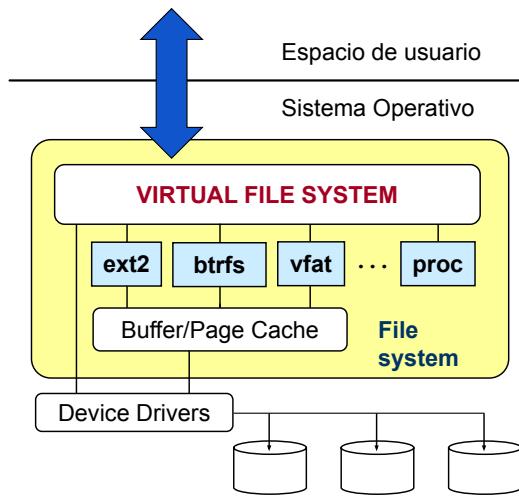
- Conjunto de tablas y estructuras que permiten gestionar los ficheros y directorios

### Tipos de Sistemas de Ficheros:

- **Basados en disco:** Residen en soportes de almacenamiento físicos como discos duros magnéticos (HDD), discos ópticos o unidades de estado sólido (SSD)
  - Ejemplos: ext2-3-4, FAT, NTFS, ISO9660, UDF, UFS, HPFS, XFS, Btrfs, ZFS...
- **Basados en red (o distribuidos):** Se utilizan para acceder a sistemas de ficheros remotos independientemente del tipo
  - Ejemplos: NFS (Network File System) y SMB
- **Basados en memoria (o pseudo):** Residen en memoria principal mientras el sistema operativo se está ejecutando
  - Ejemplos: proc, tmpfs...

**filesystems (5)**

## Estructura

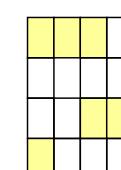
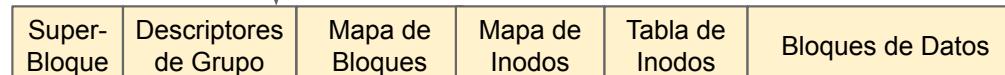
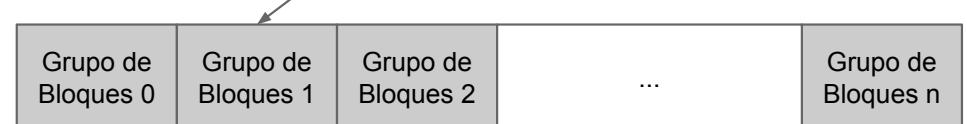


- La capa VFS establece un **enlace bien definido entre el kernel del SO y los diferentes sistemas de ficheros**
  - Proporciona las diferentes **llamadas** para la gestión de ficheros, **independientes del sistema de ficheros**
  - Permite acceder a múltiples sistemas de ficheros distintos
- Se **optimiza la entrada/salida** por medio de:
  - La **cache** de inodos y la **cache** de entradas de directorio (*dentry*) de VFS
  - La **cache** de *buffers/páginas* (*sync*)

## Estructura: Grupos de bloques

Evolución del sistema de ficheros: Minix → ext → ext2 → ext3 → ...  
Inspirado en el FFS (Fast File System) de BSD

### Estructura de ext2:



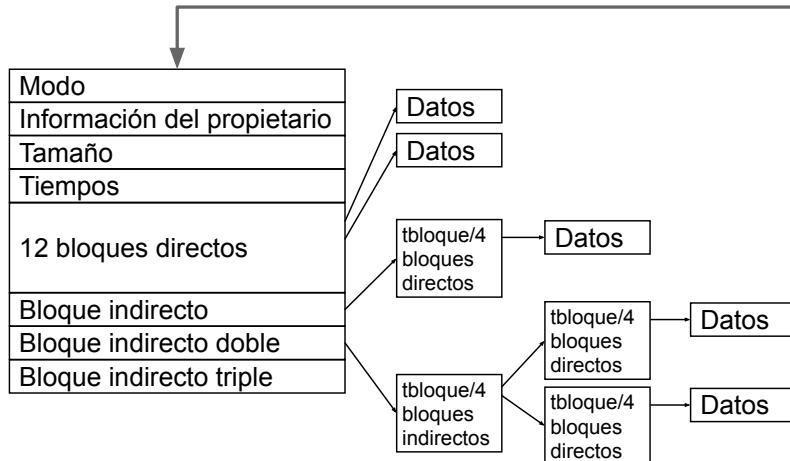
**ext2 (5)**  
**ext3 (5)**  
**ext4 (5)**

# Estructura: Inodos

## Directorio:

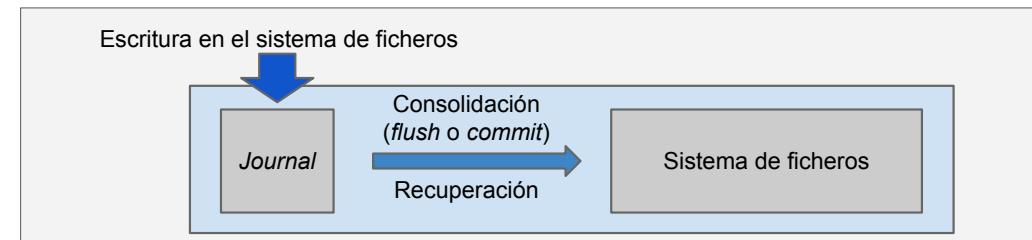
.	#555	..	#2	home	#345	vmlinuz	#654
---	------	----	----	------	------	---------	------

## Inodo:



# Journaling

- Cuando un sistema de ficheros tradicional no se apaga correctamente, el SO debe comprobar su integridad y consistencia en el siguiente arranque (utilidad `fsck`)
  - Implica recorrer toda la estructura en búsqueda de inodos huérfanos e inconsistencias, lo que puede llevar mucho tiempo en sistemas grandes
  - En ocasiones no es posible reparar automáticamente la estructura, y se debe hacer de manera manual
- Los sistemas de ficheros modernos (XFS, ext3...) incorporan un fichero, región o dispositivo especial, denominado *log* o *journal*
  - Los metadatos (inodos, mapas...) se escriben primero en el *journal*
  - En caso de fallo se usa el *journal* para devolver el FS a un estado consistente
  - La consolidación se hace periódicamente o si el *journal* supera un tamaño



# Atributos de ficheros

- Obtener el estado de un fichero:

```

int stat(const char *file_name,
         struct stat *buf);
int lstat(const char *file_name,
          struct stat *buf);
int fstat(int fd, struct stat *buf);

o stat sigue enlaces simbólicos, mientras que lstat no
o fstat recibe un descriptor de fichero abierto
o No se necesitan permisos sobre el fichero, pero sí para buscar en la ruta
o Errores:
  ■ EBADF: Descriptor no válido
  ■ ENOENT: Ruta incorrecta o nula
  ■ ENOTDIR: Componente de la ruta no es un directorio
  ■ ELOOP: Demasiados links en la búsqueda
  ■ EFAULT: Dirección no válida
  ■ EACCES: Permiso denegado
  ■ ENAMETOOLONG: Nombre de fichero muy largo
  
```

- El comando `stat` proporciona acceso a esta funcionalidad

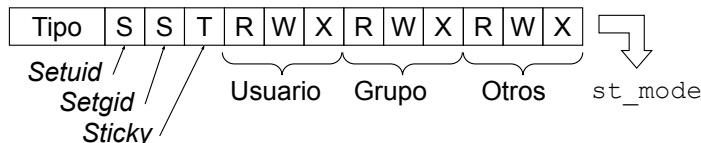
<sys/types.h>
<sys/stat.h>
SV+BSD+POSIX

```

struct stat {
    dev_t st_dev;           /* Dispositivo que lo contiene */
    ino_t st_ino;           /* Inodo */
    mode_t st_mode;         /* Protección */
    nlink_t st_link;        /* Número de enlaces rígidos */
    uid_t st_uid;           /* UID del propietario */
    gid_t st_gid;           /* GID del propietario */
    dev_t st_rdev;          /* Dispositivo, si fich. especial */
    off_t st_size;          /* Tamaño en bytes */
    unsigned long st_blksize; /* Tamaño de bloque E/S */
    unsigned long st_blocks; /* Bloques asignados */
    time_t st_atime;        /* Último acceso */
    time_t st_mtime;        /* Última modificación */
    time_t st_ctime;        /* Último cambio de estado */
}
  
```

- `st_blksize`: Tamaño de bloque para E/S de sistema de ficheros eficiente
- `st_blocks`: Número de bloques de 512 bytes asignados al fichero
- `st_atime`: Último acceso a los datos (`read, execve...`)
- `st_mtime`: Última modificación de los datos (`write, mknod...`), no del inodo
- `st_ctime`: Último cambio en el inodo (propietario, permisos, tamaño...)

## Atributos de ficheros



- Macros y *flags* (`sys/stat.h`) para comprobar el tipo de fichero y los permisos:

- Macros para **comprobar el tipo de fichero**:

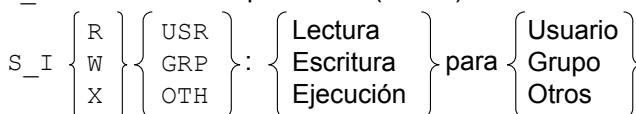
`S_ISLNK(m)`: Comprueba si es un enlace simbólico  
`S_ISREG(m)`: Comprueba si es un fichero normal  
`S_ISDIR(m)`: Comprueba si es un directorio  
`S_ISCHR(m)`: Comprueba si es un dispositivo por caracteres  
`S_ISBLK(m)`: Comprueba si es un dispositivo por bloques  
`S_ISFIFO(m)`: Comprueba si es un FIFO o pipe  
`S_ISSOCK(m)`: Comprueba si es un socket

- Flags para **comprobar permisos** (con operadores de bit | & ~ ^):

`S_IRWXU`: Permisos para el usuario (00700)

`S_IRWXG`: Permisos para el grupo (00070)

`S_IRWXO`: Permisos para otros (00007)



## Atributos de ficheros: Permisos

- Comprobar el tipo de permisos sobre un fichero:

```
int access(const char *path, int mode);
```

- El modo es una combinación de los siguientes *flags*:

- `R_OK`: El fichero existe y tenemos permisos de lectura
- `W_OK`: El fichero existe y tenemos permisos de escritura
- `X_OK`: El fichero existe y tenemos permisos de ejecución
- `F_OK`: El fichero existe

- En la comprobación de los permisos se tiene en cuenta la ruta completa

- La comprobación se realiza con los identificadores de usuario y grupo reales, a diferencia de la escritura o lectura

- La llamada fallará si alguno de los permisos no se satisface

<unistd.h>  
SV+BSD+POSIX

## Atributos de ficheros: Permisos

- Cambiar los permisos (no se puede cambiar el tipo):

```
int chmod(const char *path, mode_t mode);
int fchmod(int fd, mode_t mode);
```

- La modificación suele hacerse leyendo los permisos actuales y realizando operaciones lógicas (bit a bit) con los flags anteriores
- El EUID del proceso debe coincidir con el del propietario del fichero, o el proceso debe ser privilegiado
- Algunos errores:
  - **EIO**: Error de E/S
  - **ENOTDIR**: Elemento del PATH no existe
  - **ELOOP**: Demasiados *links* simbólicos

- El comando `chmod` proporciona acceso a esta funcionalidad

<sys/types.h>  
<sys/stat.h>  
SV+BSD+POSIX

## Creación y apertura de ficheros

- Abrir y/o crear un fichero o dispositivo:

```
int open(const char *path, int flags);
int open(const char *path, int flags,
        mode_t mode);
```

- path es la ruta del fichero o dispositivo
- flag debe indicar el modo de acceso y puede incluir otras opciones:
  - `O_RDONLY`: Acceso de solo lectura
  - `O_WRONLY`: Acceso de solo escritura
  - `O_RDWR`: Acceso de lectura y escritura
- mode indica los permisos a aplicar en caso de que se cree un nuevo fichero (con `O_CREAT`)
  - En octal (precedidos por 0 en C/C++) o como OR bit a bit de *flags*
  - Estos permisos se ven modificados por el `umask` del proceso
- Devuelve un descriptor de fichero con el puntero de acceso posicionado al principio del fichero, o -1 si ocurre un error (y establece `errno`)
  - El descriptor del fichero es el menor disponible en el sistema

<sys/types.h>  
<sys/stat.h>  
<fcntl.h>  
SV+BSD+POSIX

# Creación y apertura de ficheros

- Opciones adicionales en `flags`:
  - `O_CREAT`: Si el fichero no existe, lo crea
    - Con los permisos proporcionados en `mode` y, si no se proporciona, con un **valor arbitrario de la pila**
    - El UID y GID del nuevo fichero serán el EUID y EGID del proceso, pero si el directorio tiene el bit `setgid` activo, el GID será el GID del directorio
  - `O_EXCL`: Usado en combinación con `O_CREAT` provoca un error si el fichero existe (*Exclusively Create*)
  - `O_TRUNC`: El fichero es truncado a tamaño 0
  - `O_APPEND`: Antes de realizar cualquier escritura se posiciona el puntero de fichero a la última posición del fichero (puede corromper ficheros en NFS)
  - `O_NONBLOCK`: Abre en modo no bloqueante, en el que ni `open(2)` ni ninguna operación de E/S posterior sobre el fichero harán que el proceso espere
  - `O_SYNC`: Abre en modo síncrono, en el que las operaciones de escritura se bloquean hasta que los datos son físicamente escritos, evitando la pérdida de información en caso de fallo del sistema pero aumentando la latencia del FS

# Creación y apertura de ficheros: Permisos

- Establecer la máscara de permisos para creación de ficheros:

```
mode_t umask(mode_t mask);
```
- Establece el valor de la máscara (`umask`) del proceso a `mask & 0777`
- `umask` es usado por `open(2)`, `mkdir(2)` y otras llamadas que crean ficheros para modificar los permisos asignados a los ficheros o directorios creados
  - En concreto, los permisos indicados en `umask` son borrados del argumento `mode` con `mode & ~umask`, por ejemplo:  
$$0666 \& \sim022 = 0644$$
- El valor por defecto de `umask` es `S_IWGRP | S_IWOTH (0022)`
- Esta llamada siempre se ejecuta correctamente y devuelve la máscara anterior
- El comando interno de la `shell` `umask` proporciona acceso a esta funcionalidad

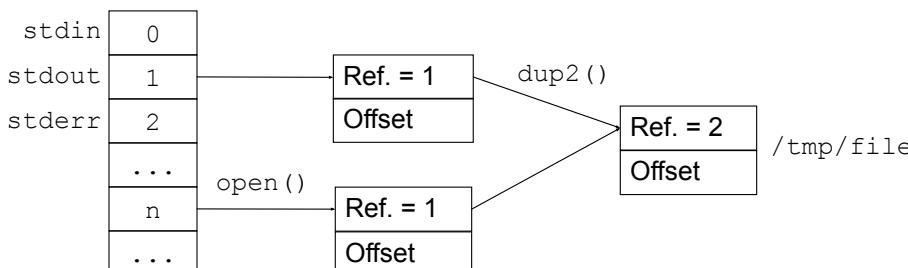
```
<sys/types.h>
<sys/stat.h>
SV+BSD+POSIX
```

# Duplicación de descriptores

- Duplicar un descriptor:

```
int dup(int oldfd);
int dup2(int oldfd, int newfd);
```
- Los dos descriptores se refieren al mismo fichero abierto, por lo que comparten cerrojos, punteros y opciones, de forma que puede intercambiarse su uso
- El descriptor devuelto por `dup()` es el menor disponible en el sistema
- Con `dup2()`, `newfd` referirá a `oldfd` y, si `newfd` está abierto, se cerrará
- Errores:
  - **EBADF**: `oldfd` no está abierto o `newfd` está fuera de rango
  - **EMFILE**: Número máximo de ficheros abiertos alcanzado

```
<unistd.h>
SV+BSD+POSIX
```



# Lectura y escritura de ficheros

- Leer, escribir, posicionar y cerrar ficheros:

```
ssize_t write(int fd, void *buffer,
              size_t count);
ssize_t read(int fd, void *buffer, size_t count);
off_t lseek(int fd, off_t offset, int where);
int close(int fd);
```
- No mezclar estas llamadas al sistema con funciones de biblioteca (ej. `fopen`, `fread`, `fwrite...` de `stdio.h`, o clases `fstream` en C++)
- La escritura de ficheros se realiza a través de la *Buffer/Page cache*, proporcionando un acceso eficiente
- Sincronizar un fichero:

```
int fsync(int fd);
```
- La llamada se bloquea hasta que el dispositivo informa de que la transferencia se ha completado

```
<unistd.h>
SV+BSD+POSIX
```

## Enlaces rígidos y simbólicos

- Crear un enlace rígido (*hard link*):

```
int link(const char *old, const char *new);
```

- Únicamente sobre ficheros en el mismo sistema de ficheros
- Si el nuevo fichero existe no será sobrescrito

- Crear un enlace simbólico (*soft link* o *symlink*):

```
int symlink(const char *old, const char *new);
```

- Entre ficheros o directorios en distintos sistemas de ficheros
- El fichero original puede no existir
- Si el nuevo fichero existe no será sobrescrito

- Leer el contenido de la ruta de un enlace simbólico:

```
int readlink(const char *path, char *b, size_t tb);
```

- El tamaño del enlace puede determinarse con `lstat(2)`
- La cadena `b` no contiene el carácter de fin de cadena

- Los comandos `ln` y `readlink` proporcionan acceso a estas funcionalidades

<unistd.h>
SV+BSD+POSIX

## Borrado de ficheros

- Eliminar un nombre de fichero y posiblemente el fichero al que se refiere:

```
int unlink(const char *name);
```

- Borra la entrada del directorio y decrementa el número de enlaces en el inodo
- Cuando el número de enlaces llega a 0 y no hay ningún proceso que mantenga abierto el fichero, este se elimina, devolviendo el espacio al sistema
- El fichero (fifo, socket, dispositivo) permanecerá en el sistema mientras que exista un proceso que lo mantenga abierto
- Un fichero en un directorio con el bit `sticky` activo solo puede ser borrado por el propietario del fichero o del directorio

<unistd.h>
SV+BSD+POSIX

- El comando `rm` proporciona acceso a esta funcionalidad

## Cerrojos de ficheros

- Crear, comprobar y eliminar un cerrojo POSIX:

```
int lockf(int fd, int cmd, off_t len);
```

- `fd` es un descriptor de fichero abierto para escritura
- `cmd` es una de las siguientes operaciones:
  - `F_LOCK`: Bloquea la región especificada del fichero, esperando hasta que un cerrojo incompatible previo se libere
  - `F_TLOCK`: Como `F_LOCK`, pero la llamada no espera, sino que devuelve -1 con `errno=EAGAIN` o `EACCES` si la región está bloqueada
  - `F_ULOCK`: Desbloquea la región indicada del fichero
  - `F_TEST`: Devuelve 0 si la región está desbloqueada o bloqueada por el proceso, o -1 con `errno=EAGAIN` o `EACCES` si la bloquea otro proceso
- `len` especifica la región (relativa a `pos`, que es la posición actual):
  - `len = 0, pos..infinito` (fin de fichero actual y sucesivos)
  - `len > 0, pos..pos+len-1`
  - `len < 0, pos-len..pos-1`
- Los cerrojos son consultivos (`read(2)` y `write(2)` no comprueban su existencia), por lo que solo son útiles entre procesos que cooperan

<unistd.h>
SV+POSIX

## Acceso a directorios

- Abrir un directorio indicado por `name`:

```
DIR *opendir(const char *name);
```

- Devuelve un puntero al flujo de directorio, posicionado en la primera entrada del directorio
- El tipo de datos `DIR` se usa de forma similar al tipo `FILE` especificado por la biblioteca de entrada salida estándar

<sys/types.h>
<dirent.h>
SV+BSD+POSIX

- Leer entradas de un directorio:

```
struct dirent *readdir(DIR *dir);
```

- La función retorna una estructura `dirent` que apunta a la siguiente entrada en el directorio, y `NULL` cuando llega al final u ocurre un error
- El único campo contemplado por el estándar POSIX es `d_name`, de longitud variable (menor que `NAME_MAX`)

- Cerrar el directorio definido por el descriptor, haciéndolo inaccesible a subsecuentes llamadas:

```
int closedir(DIR *dir);
```

# Creación y borrado de directorios

# Ejemplos de Preguntas Teóricas

- Crear un directorio:

```
int mkdir(const char *path, mode_t mode);
```

- mode especifica los permisos para el nuevo directorio (modificados por umask)
- El UID y GID del nuevo directorio serán el EUID y EGID del proceso, pero si el directorio padre tiene el bit setgid activo, el GID será el GID del directorio

<sys/stat.h>
<sys/types.h>
SV+BSD+POSIX

- Eliminar un directorio:

```
int rmdir(const char *path);
```

- Cambiar el nombre o ubicación de un fichero o directorio:

```
int rename(const char *old,  
          const char *new);
```

- Si new existe se elimina y si es un directorio ha de estar vacío
- old y new han de ser del mismo tipo y estar en el mismo sistema de ficheros
- Si old es un enlace simbólico será renombrado, si lo es new será sobrescrito

<unistd.h>
SV+BSD+POSIX
<stdio.h>
BSD+POSIX

- Un fichero en un directorio con el bit sticky activo solo puede ser borrado o renombrado por el propietario del fichero o del directorio

- Los comandos mkdir, rmdir y mv proporcionan acceso a estas funcionalidades



## AMPLIACIÓN DE SISTEMAS OPERATIVOS Y REDES

Grado en Ingeniería Informática / Doble Grado  
Universidad Complutense de Madrid

## Material adicional

¿Qué máscara (umask) debería fijar el proceso para que los ficheros se creen con los permisos simbólicos rw- -w- r--, considerando que open(2) especifica los permisos 0666?

- 0024.
- 0042.
- 0624.

¿Cuál de las siguientes afirmaciones sobre dup(2) y dup2(2) es cierta?

- Solo se pueden duplicar descriptores asociados a ficheros regulares.
- Los descriptores 0 (stdin), 1 (stdout) y 2 (stderr) no se pueden duplicar.
- Despues de duplicar un descriptor, se puede cerrar.

¿Qué diferencias hay entre un enlace rígido y uno simbólico a un fichero?

- El enlace rígido tiene el mismo inodo que el fichero original y el simbólico, uno distinto.
- El enlace simbólico incrementa el número de enlaces del inodo y el simbólico no.
- El enlace rígido se puede hacer a cualquier fichero y el simbólico solo dentro del mismo sistema de ficheros.

## Control de ficheros

- Manipular un descriptor de fichero:

```
int fcntl(int fd, int cmd);  
int fcntl(int fd, int cmd, long arg);
```

<unistd.h>
<fcntl.h>
SV+BSD+POSIX

- cmd determina la operación que se realizará sobre el fichero:
  - F\_DUPFD: Duplica el descriptor como dup(2)
  - F\_GETFD: Obtiene los flags del descriptor (FD\_CLOEXEC)
  - F\_SETFD: Fija los flags del descriptor al valor especificado en arg
  - F\_GETFL: Obtiene los flags del fichero que se fijaron con open(2)
  - F\_SETFL: Fija algunos flags del fichero (por ejemplo O\_APPEND, O\_NONBLOCK o O\_ASYNC) al valor especificado en arg

# Control de ficheros: Cerrojos

# Control de ficheros: Cerrojos

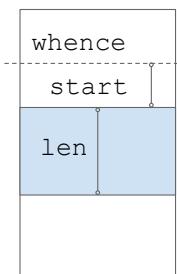
- Bloquear regiones de un fichero:

```
int fcntl(int fd, int cmd, struct flock *lock);

struct flock {
    short int l_type; /* F_RDLCK, F_WRLCK o F_UNLCK */
    short int l_whence; /* SEEK_SET, SEEK_CUR o SEEK_END */
    off_t l_start; /* Inicio de la región bloqueada */
    off_t l_len; /* Tamaño de la región (0=hasta EOF) */
    pid_t l_pid; /* Proceso que bloquea (F_GETLK) */
}
```

- Tipos de cerrojos:

- **De lectura o compartido (F\_RDLCK)**: El proceso está leyendo el área bloqueada por lo que no puede ser modificada
  - Pueden establecerse varios sobre una misma región
- **De escritura o exclusivo (F\_WRLCK)**: El proceso está escribiendo, por lo que ningún otro debe leer o escribir del área bloqueada
  - Solo puede haber uno



- cmd determina la operación que se realizará sobre el cerrojo:
  - F\_GETLK: Comprueba si se puede activar el cerrojo descrito en lock
    - Si se puede activar, establece el campo l\_type de lock a F\_UNLCK
    - Si no, devuelve en lock los detalles de uno de los cerrojos que lo impiden, incluyendo el PID del proceso que lo mantiene
  - F\_SETLK: Activa (si l\_type es F\_WRLCK o F\_RDLCK) o libera (si l\_type es F\_UNLCK) el cerrojo descrito por lock
    - Si hay un cerrojo incompatible, devuelve -1 con errno=EAGAIN
  - F\_SETLKW: Igual que F\_SETLK, pero
    - Si hay un cerrojo incompatible, espera a que sea liberado
- Los cerrojos activos pueden consultarse en /proc/locks
- flock(2) y flock(1) permiten gestionar cerrojos BSD (no POSIX)



## TEMA 2.3. Gestión de Procesos

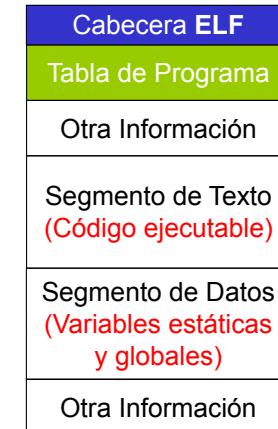
### PROFESORES:

Rubén Santiago Montero  
Eduardo Huedo Cuesta  
Luis M. Costero Valero

## Estructura de un Programa

- Un programa es un conjunto de instrucciones máquina y datos, almacenados en una imagen ejecutable en disco. Es una entidad pasiva.

### Executable and Linking Format (ELF)



### Organización y atributos

```

typedef struct {
    Elf32_Half e_type;
    Elf32_Half e_machine;
    ...
} Elf32_Ehdr;
    
```

ET\_REL: Relocatable object  
ET\_EXEC: Executable  
ET\_DYN: Shared object  
ET\_CORE: Core  
EM\_386, EM\_X86\_64,  
EM\_IA\_64, EM\_SPARC...

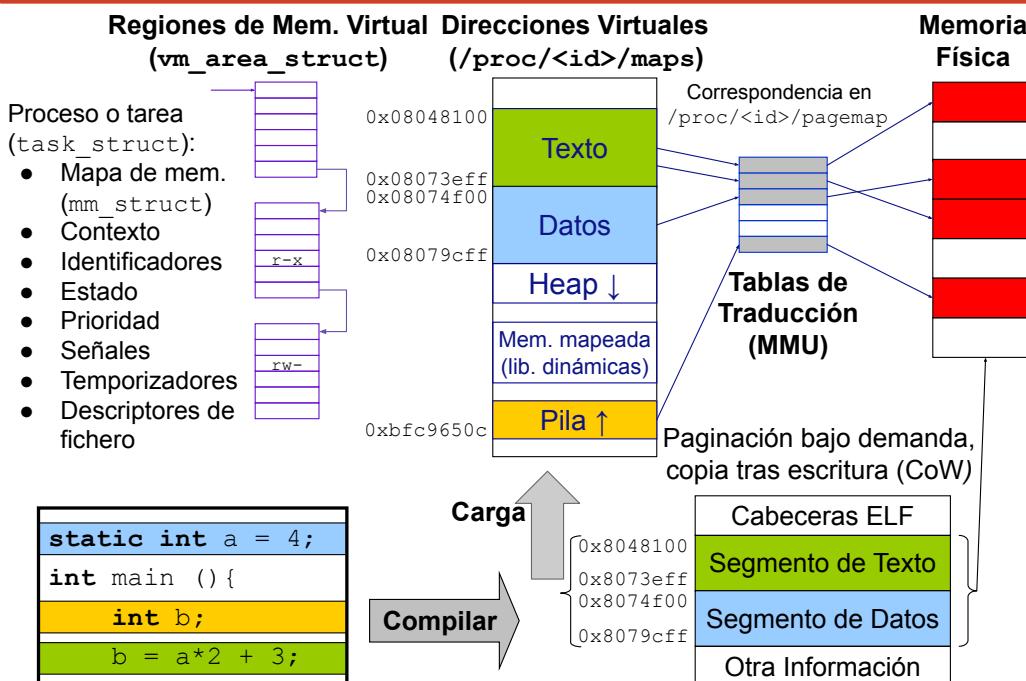
### Información para ejecución

```

typedef struct {
    Elf32_Word p_type;
    Elf32_Addr p_vaddr;
    Elf32_Word p_filesz;
    Elf32_Word p_memsz;
    ...
} Elf32_Phdr;
    
```

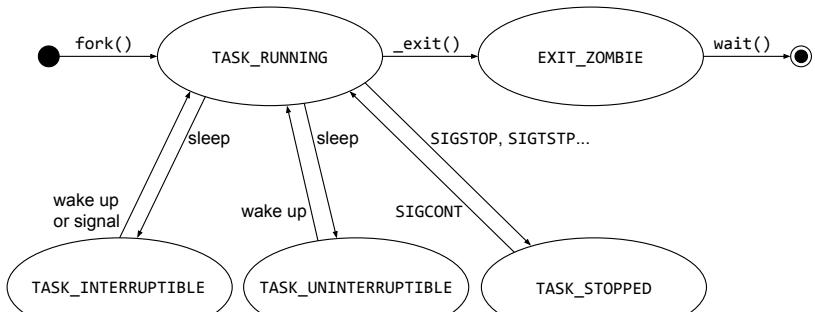
PT\_PHDR: Program header  
PT\_LOAD: Loadable segment  
PT\_DYNAMIC: Dynamic linking  
Dirección virtual del segmento

## Estructura de un Proceso



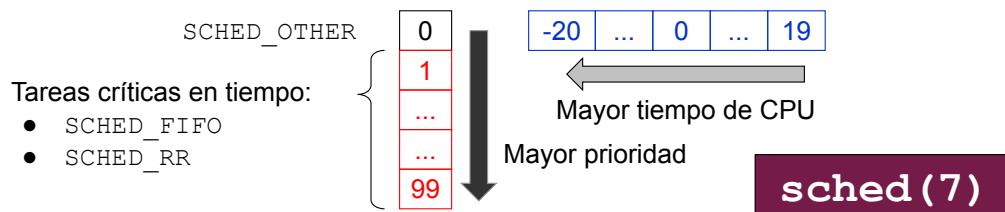
## Estados de un Proceso

- El comando ps muestra la lista de procesos, sus identificadores y sus atributos, incluyendo el estado:
  - R (TASK\_RUNNING): en ejecución o preparado (en la cola de ejecución)
  - D (TASK\_UNINTERRUPTIBLE): bloqueado (normalmente por E/S)
  - S (TASK\_INTERRUPTIBLE): esperando (a que un evento se complete)
  - T (TASK\_STOPPED): parado (por una señal)
  - Z (EXIT\_ZOMBIE): muerto (ya no existe, pero deja su entrada en la tabla de procesos para que el proceso padre recoja su estado de salida)



# Planificador

- Componente del núcleo que determina el orden de ejecución de las tareas en función de su prioridad y de la clase de planificación
  - Es expropiativo (una tarea de mayor prioridad siempre expropiará a otra de menor prioridad en ejecución), y la política de planificación solo determina el orden de ejecución de la lista de tareas preparadas con igual prioridad
- **Políticas de planificación** (ver /usr/include/bits/sched.h)
  - **SCHED\_OTHER**: Política estándar de tiempo compartido con prioridad 0, que considera el valor de *nice* (entre -20 y 19, 0 por defecto) para repartir la CPU
  - **SCHED\_FIFO**: Política de tiempo real FIFO con prioridades entre 1 y 99
    - Una tarea de esta política se ejecutará hasta que se bloquee por E/S, sea expropiada por una tarea con mayor prioridad o ceda la CPU
  - **SCHED\_RR**: Como la anterior, pero los tareas con igual prioridad se ejecutan por turnos (*round-robin*) durante un *cuento* de tiempo máximo



# Planificador

- Obtener y establecer la prioridad de planificación:

```
int sched_getparam(pid_t pid, struct sched_param *p);
int sched_setparam(pid_t pid, const struct sched_param *p)
```

  - *pid* es un PID (0 para el proceso actual)
  - *p* para obtener o establecer la nueva prioridad
- Consultar los rangos de prioridades:

```
int sched_get_priority_max(int policy);
int sched_get_priority_min(int policy);
```

  - *policy* selecciona la política de planificación
- El comando **chrt** (*change real-time*) ofrece acceso a esta funcionalidad

# Planificador

- Consultar y establecer la política de planificación y establecer la prioridad:

```
int sched_getscheduler(pid_t pid);
int sched_setscheduler(pid_t pid, int policy,
                      const struct sched_param *p);

struct sched_param {
    int sched_priority;
    ...
};
```

  - *pid* es un PID (un valor 0 se refiere al proceso actual)
  - *policy* selecciona la política de planificación
  - *p* establece la nueva prioridad
- Las llamadas afectan realmente al *thread* (el planificador maneja *threads* o tareas)
  - Todas las llamadas tienen su equivalente *pthread\_\**
- Las llamadas *fork()* heredan los atributos de planificación

<sched.h>

POSIX

# Planificador

- Obtener y establecer el valor de *nice* de un proceso:

```
int getpriority(int which, int who);
int setpriority(int which, int who, int prio);
```

  - *which* puede ser PRIO\_PROCESS, PRIO\_PGRP o PRIO\_USER
  - *who* es un PID, un PGID o un UID, respectivamente
    - 0 indica el proceso actual, el grupo de procesos del proceso actual o el UID real del proceso actual, respectivamente
  - *prio* es el nuevo valor de *nice* entre -20 y 19
    - Valores menores representan una mayor porción de CPU
- Los comandos **nice** y **renice** permiten acceder a esta funcionalidad

<sys/time.h>

<sys/resource.h>

SV+BSD

## Identificadores de un Proceso

- Cada proceso tiene un identificador único (Process ID, PID) y, además, registra el proceso que lo creó (Parent Process ID, PPID)
- Obtener los identificadores de un proceso:

```
pid_t getpid(void);  
pid_t getppid(void);
```

<unistd.h>  
SV+BSD+POSIX

credentials (7)

## Identificadores de un Proceso: Grupos

- Los procesos pertenecen a un grupo de procesos, con un PGID (Process Group ID) igual al PID del proceso líder del grupo
  - Su principal uso es la distribución de señales
- Obtener o establecer el identificador del grupo de procesos:  

```
pid_t getpgid(pid_t pid);  
int setpgid(pid_t pid, pid_t pgid);
```

  - Si `pid` es 0, se refiere al proceso que hace la llamada
  - Si `pgid` es 0, se usa el PID del proceso indicado en `pid`

<unistd.h>  
POSIX

## Identificadores de un Proceso: Sesiones

- Los grupos de procesos se pueden agrupar en sesiones, con un SID (Session ID)
  - Se usan para gestionar el acceso al sistema
- Un proceso de *login* crea una sesión y abre un terminal de control
  - Todos los procesos y grupos de procesos del usuario pertenecen a esa sesión y comparten el terminal
  - En la desconexión, se envía la señal `SIGHUP` a todos los procesos de la sesión
- Un proceso puede crear una sesión si no es el líder de un grupo de procesos
  - Para asegurarse de que no es el líder, suele hacer `fork(2)` primero
  - El proceso es el líder de la sesión (su SID se establece a su PID) y de un nuevo grupo de procesos en la sesión (su PGID se establece a su PID)
  - Inicialmente, la sesión no tiene terminal de control
- Obtener el identificador de sesión:  

```
pid_t getsid(pid_t pid);
```
- Crear una nueva sesión y un nuevo grupo de procesos:  

```
pid_t setsid(void);
```
- El comando `setsid` permite crear una nueva sesión (y un nuevo grupo asociado)

<unistd.h>  
SV+POSIX

## Directorio de Trabajo

- Es el directorio usado para resolver toda ruta relativa en el proceso
- Obtener la **ruta absoluta** del directorio de trabajo:  

```
char *getcwd(char *buffer, size_t size);
```

  - La ruta se copia en `buffer` de tamaño `size`
  - Si el tamaño de la ruta, incluyendo el carácter '\0' de fin de cadena, excede `size` bytes, la función devuelve `NULL` y establece `errno=ERANGE`
- Cambiar el directorio de trabajo de un proceso:  

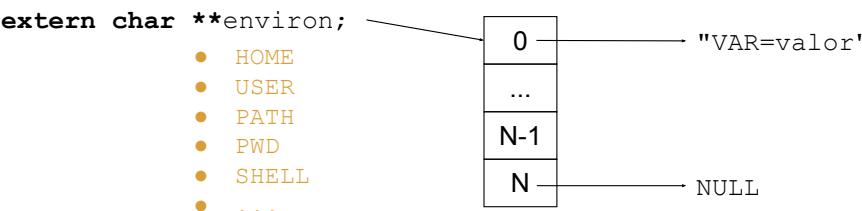
```
int chdir(const char *path);
```
- El comando `pwd` y los comandos internos de la *shell* `pwd` y `cd` proporcionan acceso a esta funcionalidad

<unistd.h>  
POSIX

## Entorno

## Creación de Procesos

- Los procesos se ejecutan en un determinado entorno, que en general se hereda del proceso padre
  - Muchas aplicaciones limitan o controlan el entorno que pasan a los procesos o la forma en que inicializan su entorno, ej. `sudo` o la `shell`
- El entorno es un conjunto de cadenas de caracteres en la forma "VARIABLE=valor"
  - Por convenio, las variables de entorno están en mayúsculas



- Obtener, establecer o eliminar variables de entorno:

```
char *getenv(const char *name);
int setenv(const char *name, const char *value, int overwrite);
int unsetenv(const char *name);
```

- El comando interno de la `shell` `export` y el comando `env` proporcionan acceso a esta funcionalidad

<unistd.h>

SV+POSIX+BSD

- Crear un proceso hijo:

```
pid_t fork(void);
```

- La función devuelve:
  - 0: Ejecutando el hijo
  - >0: Ejecutando el padre, el valor es el PID del hijo
  - 1: Fallo
- El nuevo proceso ejecuta el mismo código que el proceso padre y recibe una copia de los descriptores de los ficheros abiertos por el padre

<stdlib.h>

SV+BSD+POSIX

## Creación de Procesos

```
int main() {
    pid_t pid;
    pid = fork();
    switch (pid) {
        case -1:
            perror("fork");
            exit(1);
        case 0:
            printf("Hijo %i (padre: %i)\n", getpid(), getppid());
            break;
        default:
            printf("Padre %i (hijo: %i)\n", getpid(), pid);
            break;
    }
    return 0;
}
```

## Finalización de un Proceso

- Un proceso puede finalizar por dos motivos:
  - Voluntariamente, llamando a `exit` (o `return` desde `main()`)
  - Al recibir una señal (hay múltiples causas)
- Terminar el proceso:

```
void _exit(int status);
```

  - `status` es el estado de salida, que debe ser un número menor que 255
    - Por convenio, 0 (`EXIT_SUCCESS`) significa éxito y 1 (`EXIT_FAILURE`) significa error
    - Nunca devolver `errno` ni -1
    - Accesible en la `shell` vía `$?` o en el proceso padre vía `wait(2)`
  - Cualquier descriptor de fichero abierto por el proceso se cierra
  - Cualquier hijo del proceso se vuelve huérfano y es heredado por el proceso 1 (`init` o `systemd`)
  - El proceso padre recibe una señal `SIGCHLD`

## Finalización de un Proceso

- Esperar la finalización (o cambio de estado) de un proceso hijo:

```
pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);
```

- pid especifica a qué procesos hijo esperar:
  - > 0 indica uno con identificador pid
  - 0 indica uno del grupo de procesos del padre
  - -1 indica cualquiera (igual que wait())
  - < -1 indica el que su PGID es -pid
- options es una OR de las siguientes opciones:
  - WNOHANG: retorna sin esperar si no hay hijos que hayan terminado
  - WUNTRACED: retorna si el proceso ha sido detenido
  - WCONTINUED: retorna si un hijo detenido ha sido reanudado
- status contiene información de estado, que puede consultarse con macros:
  - WIFEXITED(s) indica si el hijo terminó normalmente vía exit() y, en ese caso, WEXITSTATUS(s) devuelve el estado de salida
  - WIFSIGNALED(s) indica si el hijo terminó al recibir una señal y, en ese caso, WTERMSIG(s) devuelve el número de la señal recibida
- Devuelve el PID del hijo terminado o -1 en caso de error
- Un proceso que termina pero no ha sido esperado se convierte en zombi

<sys/types.h>  
<sys/wait.h>  
SV+POSIX+BSD

## Ejecución de Programas

- Ejecutar un programa:

```
int execve(const char *filename, char *const argv[],
           char *const envp[]);
```

- Sustituye la imagen del proceso actual por una nueva
- El primer elemento de argv debe ser el nombre de fichero del programa (ejecutable binario o script) y el último ha de ser NULL

- Las siguientes funciones usan la llamada execve(2):

```
int execl(const char *path, const char *a0, ...);
int execlp(const char *file, const char *a0, ...);
int execle(const char *path, const char *a0, ...,
            char *const envp[]);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
```

	Ruta absoluta	Ruta relativa	Nuevo entorno
Lista de argumentos	execl()	execlp()	execle()
Vector de argumentos	execv()	execvp()	execve()

## Ejecución de Programas

- Ejecutar un comando de la shell:

```
int system(const char *command);
```

- Usa fork(2) para crear un proceso hijo que ejecute el comando de la shell especificado en command usando execl(3) como:

```
execl("/bin/sh", "sh", "-c", command, (char *) 0);
```
- La llamada retorna cuando termina la ejecución del comando (salvo si se ejecuta en segundo plano)
- Devuelve el código de finalización del comando, obtenido con waitpid(2), o -1 en caso de error

<stdlib.h>  
ANSI C+POSIX

## Límites de Recursos

- Obtener y establecer los límites del proceso:

```
int getrlimit(int resource, struct rlimit *rlim);
int setrlimit(int resource, const struct rlimit *rlim);
```

```
struct rlimit{
    int rlim_cur; /* Límite actual */
    int rlim_max; /* Valor máximo */
};
```

- resource puede ser
  - RLIMIT\_CPU: Max. tiempo de CPU (segundos)
  - RLIMIT\_FSIZE: Max. tamaño de fichero (bytes)
  - RLIMIT\_DATA: Max. tamaño del heap (bytes)
  - RLIMIT\_STACK: Max. tamaño de pila (bytes)
  - RLIMIT\_CORE: Max. tamaño de fichero core (bytes)
  - RLIMIT\_NPROC: Max. número de procesos
  - RLIMIT\_NOFILE: Max. número de descriptores de fichero
- rlim especifica el límite (el valor RLIM\_INFINITY indica ilimitado)

- El comando interno de la shell ulimit proporciona acceso a esta llamada

<sys/time.h>  
<sys/resource.h>  
SV+BSD



- Obtener el uso de recursos:

```
int getrusage(int who, struct rusage *usage);
struct rusage {
    struct timeval ru_utime; /* t. CPU en modo usuario */
    struct timeval ru_stime; /* t. CPU en modo sistema */
    long ru_maxrss; /* RSS máximo */
    long ru_minflt; /* páginas reclamadas */
    long ru_majflt; /* fallos de página */
    long ru_inblock; /* ops. de entrada de bloques */
    long ru_oublock; /* ops. de salida de bloques */
    ...
    /* ver man getrusage */
}
```

- who puede ser
  - RUSAGE\_SELF: por el proceso (todos sus *threads*)
  - RUSAGE\_CHILDREN: por todos los hijos del proceso
  - RUSAGE\_THREAD: por el *thread*

- El comando `time -v` proporciona esta información (`time` es también una palabra reservada de la *shell*)

## Señales

## Señales

- Las señales son **interrupciones software**, que informan a un proceso de la ocurrencia de un evento de forma **asíncrona**
  - Las genera un proceso o el núcleo del sistema
- Las opciones en la ocurrencia de un evento son:
  - Bloquear la señal
  - Ignorar la señal
  - Realizar la acción por defecto asociada a la señal, que en general consiste en terminar la ejecución del proceso
  - Capturar la señal con un manejador, que es una función definida por el programador que se invoca automáticamente al recibir la señal
- Tipos de señales:
  - Terminación de procesos
  - Excepciones
  - Llamada de sistema
  - Generadas por proceso
  - Interacción con el terminal
  - Traza de proceso
  - Fuertemente dependientes del sistema (consultar `signal.h`)

## Señales: System V (Ejemplos)

- **SIGHUP**: Desconexión de terminal (**F**, terminar proceso)
- **SIGINT**: Interrupción. Se puede generar con `Ctrl+C` (**F**)
- **SIGQUIT**: Finalización. Se puede generar con `Ctrl+\` (**F** y **C**, volcado de mem.)
- **SIGSTOP**: Parar proceso. No se puede capturar, bloquear o ignorar (**P**, parar)
- **SIGTSTP**: Parar proceso. Se puede generar con `Ctrl+Z` (**P**)
- **SIGCONT**: Reanudar proceso parado (continuar)
- **SIGILL**: Instrucción ilegal (punteros a funciones mal gestionados) (**F** y **C**)
- **SIGTRAP**: Ejecución paso a paso, enviada después de cada instrucción (**F** y **C**)
- **SIGKILL** (9): Terminación brusca. No se puede capturar, bloquear o ignorar (**F**)
- **SIGBUS**: Error de acceso a memoria (alineación o dirección no válida) (**F** y **C**)
- **SIGSEGV**: Violación de segmento de *datos* (**F** y **C**)
- **SIGPIPE**: Intento de escritura en un tubería sin lectores (**F**)
- **SIGALRM**: Despertador, contador a 0 (**F**)
- **SIGTERM**: Terminar proceso (**F**)
- **SIGUSR1, SIGUSR2**: Señales de usuario (**F**)
- **SIGCHLD**: Terminación del proceso hijo (**I**, ignorar)
- **SIGURG**: Recepción de datos urgentes en socket (**I**)

signal (7)

## Señales: Envío

- Enviar una señal a un proceso:

```
int kill(pid_t pid, int signal);
```

<signal.h>  
SV+BSD+POSIX

- pid identifica el proceso que recibirá la señal:
  - >0: Es el identificador del proceso
  - 0: Se envía a todos los procesos del grupo
  - 1: Se envía a todos los procesos (de mayor a menor), excepto el 1
  - <-1: Se envía a todos los procesos del grupo con PGID igual a -pid
- signal es la señal que se enviará (si es 0, se simula el envío)

- El comando `kill`, que también es un comando interno de la *shell*, proporciona acceso a esta llamada

- Llamadas equivalentes:

```
int raise(int signal);
int abort(void);
```

- `raise(signal) ⇒ kill(getpid(), signal)`
- `abort() ⇒ kill(getpid(), SIGABRT)`

<signal.h>  
ANSI-C  
<stdlib.h>  
SV+BSD+POSIX

## Señales: Ejemplo de Envío

```
#include <signal.h>
#include <unistd.h>

int main()
{
    kill(getpid(), SIGABRT);
    return 0;
}
```

```
> ./abort_self
Aborted (core dumped)
```

## Señales: Conjuntos de Señales

- Las señales se agrupan en conjuntos de señales POSIX para definir máscaras de señales
  - Tipo opaco `sigset_t` que depende del sistema
  - Implementado como mapa de bits (ver `/usr/include/bits/sigset.h`)
- Operaciones con conjuntos de señales POSIX:

```
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signal);
int sigdelset(sigset_t *set, int signal);
int sigismember(sigset_t *set, int signal);
```

<signal.h>  
POSIX

- `sigemptyset()` inicializa un conjunto como vacío, excluyendo todas las señales
- `sigfillset()` inicializa un conjunto como lleno, incluyendo todas las señales
- `sigaddset()` añade una señal a un conjunto
- `sigdelset()` elimina una señal de un conjunto
- `sigismember()` comprueba si una señal pertenece a un conjunto

## Señales: Bloqueo

- La máscara de señales es el conjunto de señales bloqueadas (por ejemplo, para proteger regiones de código)
- Consultar y establecer las señales bloqueadas:

```
int sigprocmask(int how, const sigset_t *set,
                sigset_t *oset);
```

- how define el comportamiento:
  - `SIG_BLOCK`: Añade el conjunto set al conjunto de señales actualmente bloqueadas ("OR")
  - `SIG_UNBLOCK`: Elimina el conjunto set del conjunto de señales bloqueadas (puede desbloquearse una señal que no estuviera bloqueada)
  - `SIG_SETMASK`: Reemplaza el conjunto de señales actuales por set
- oset almacena el conjunto previo de señales bloqueadas (distinto de NULL)
- Comprobar señales pendientes:

```
int sigpending(const sigset_t *set);
```

- set es el conjunto de señales pendientes
- Usar `sigismember()` para determinar la señal y `sigprocmask()` para desbloquearla y tratarla

<signal.h>

POSIX

## Señales: Ejemplo de Bloqueo

```
#include <stdlib.h>
#include <signal.h>

int main() {
    sigset(SIG_BLOCK, SIG_BLOCK, SIG_BLOCK);

    /* Código protegido */

    sigset(SIG_BLOCK, SIG_BLOCK, SIG_BLOCK);
}
```

## Señales: Captura

- Es posible modificar la acción por defecto realizada por un proceso al recibir una señal definiendo una función manejadora de la señal (*handler*)
- Obtener y establecer la acción asociada a una señal:

```
int sigaction(int signal,
              const struct sigaction *act,
              struct sigaction *oldact);

struct sigaction {
    void (*sa_handler)(int);
    sigset(SIG_BLOCK, sa_mask);
    int sa_flags;
    ...
}
```

<signal.h>

POSIX

- *signal* especifica la señal (excepto SIGKILL y SIGSTOP)
- *act* contiene el nuevo manejador para la señal (puede ser NULL)
- *oldact* almacenará el antiguo controlador de la señal (puede ser NULL)

## Señales: Captura

- Campos de la estructura *sigaction*:
  - *sa\_handler* es el nuevo manejador para la señal. Su valor puede ser:
    - SIG\_DFL para la acción por defecto
    - SIG\_IGN para ignorar la señal
    - Un puntero a una función:

```
void handler(int signal);
```
  - *sa\_mask* es el conjunto de señales que serán bloqueadas durante el tratamiento de la señal
    - Además, por defecto se bloquea la señal en cuestión
  - *sa\_flags* modifica el comportamiento del proceso de gestión de la señal:
    - SA\_NODEFER no bloquea la señal que se está tratando
    - SA\_RESTART reinicia ciertas llamadas al sistema interrumpidas para compatibilidad con BSD (en otro caso, fallan con *errno*=EINTR)
    - SA\_RESETHAND restaura el manejador por defecto tras tratar la señal
    - SA\_SIGINFO usa una función para tratar la señal con argumentos adicionales (campo *sa\_sigaction*)

## Señales: Captura

- La ejecución del proceso se interrumpe y se llama al manejador
  - Cuando el manejador termina, se restaura la ejecución en el punto donde se produjo la señal
- Hay que tomar algunas precauciones en el manejador:
  - Declarar las variables globales como *volatile*
  - No usar **funciones no reentrantes**, como *malloc*, *free* o funciones de la biblioteca *stdio*
  - Guardar y restaurar el valor de *errno* si llama a alguna función que pueda modificarlo
- Como regla general, hacer lo menos posible en el manejador
  - Normalmente, fijar algún *flag* y salir
- Tener en cuenta siempre que las señales son **asíncronas**

## Señales: Espera

- Esperar la ocurrencia de una determinada señal, suspendiendo la ejecución del proceso:

```
int sigsuspend(const sigset_t *set);
```

- La máscara de señales bloqueadas se sustituye temporalmente por el conjunto set, el proceso se suspende hasta que **una señal que no esté en la máscara** se produzca
- Cuando se recibe la señal se **ejecuta el manejador** asociado a la señal y continúa la ejecución del proceso, restaurando la **máscara original**
- Siempre devuelve -1 y, normalmente, establece errno a EINTR

- Alternativamente, suspender un proceso de forma más sencilla:

```
unsigned int sleep(unsigned int segundos);
int pause(void);
```

- Suspenden la ejecución durante los segundos especificados o indefinidamente, respectivamente, o bien, hasta recibir una señal que deba ser tratada

<signal.h>
POSIX

## Señales: Alarmas y Temporizadores

- Fijar una alarma:

```
unsigned int alarm(unsigned int secs);
```

- Se programa el temporizador ITIMER\_REAL para generar una señal SIGALRM en secs segundos (si es cero, no se planifica ninguna nueva alarma)
  - Cualquier alarma programada previamente se cancela
  - Debe instalarse antes un manejador
- Devuelve el valor de segundos restantes para que se produzca el final de la cuenta (0 si no hay ninguna fijada)
- No mezclar con sleep(3) o cualquier otra función que use el mismo temporizador, como setitimer(2)
- No se heredan con fork(2), pero sí se mantienen tras execve(2)

<unistd.h>
SV+BSD+POSIX

## Señales: Alarmas y Temporizadores

- Consultar o fijar alarmas asociadas a otros temporizadores:

```
int getitimer(int which, struct itimerval *curr_value);
int setitimer(int which, struct itimerval *new_value,
             struct itimerval *old_value);

struct itimerval {
    struct timeval it_interval; /* Intervalo */
    struct timeval it_value;   /* Tiempo que queda */
}
```

<sys/time.h>
SV+BSD



## AMPLIACIÓN DE SISTEMAS OPERATIVOS Y REDES

Grado en Ingeniería Informática / Doble Grado  
Universidad Complutense de Madrid

## Comunicación entre Procesos. Tuberías

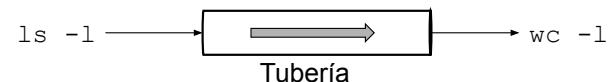
# Introducción

# Tuberías sin Nombre

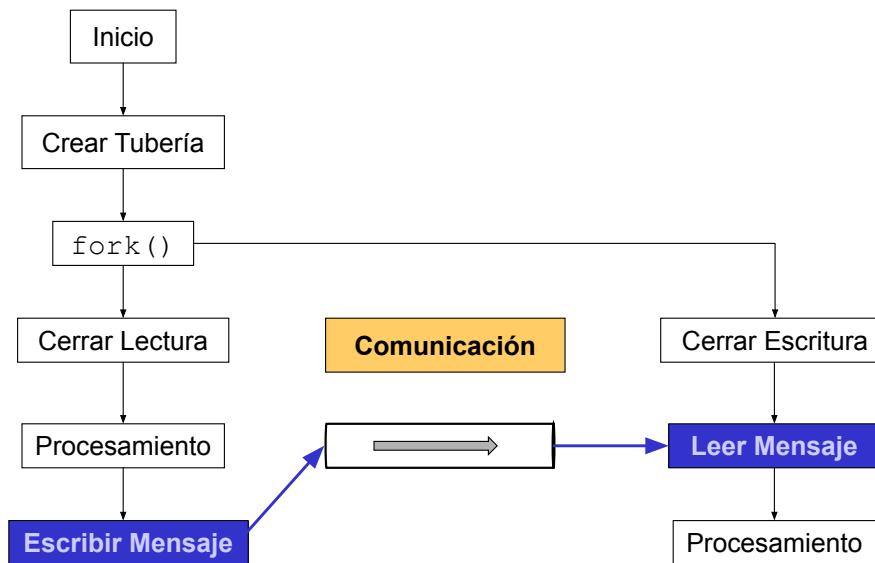
- Mecanismos de sincronización:
  - Mismo sistema
    - Señales (Tema 2.3)
    - Ficheros con cerrojos (Tema 2.2)
    - Mutex y variables de condición (solo para threads de un proceso)
    - Semáforos (System V IPC)
    - Colas de mensajes (System V IPC)
  - Distintos sistemas
    - Basados en sockets (Tema 2.4)
- Compartición de datos entre procesos:
  - Mismo sistema
    - Memoria compartida (System V IPC)
    - Tuberías sin nombre o *pipes* (Tema 2.3)
    - Tuberías con nombre o FIFOs (Tema 2.3)
    - Colas de mensajes (System V IPC)
    - Basados en ficheros (Tema 2.2)
  - Distintos sistemas
    - Basados en sockets (Tema 2.4)

- Proporcionan un canal de comunicación unidireccional entre procesos
- El sistema las **trata** a todos los efectos **como ficheros**:
  - i-nodo
  - Descriptores
  - Tabla de ficheros del sistema y proceso
  - Operaciones de E/S típicas
  - Heredadas de padres a hijos
- **Sincronización** realizada por parte del **núcleo**
- Acceso tipo **FIFO** (*first-in-first-out*)
- La tubería **reside en memoria principal**

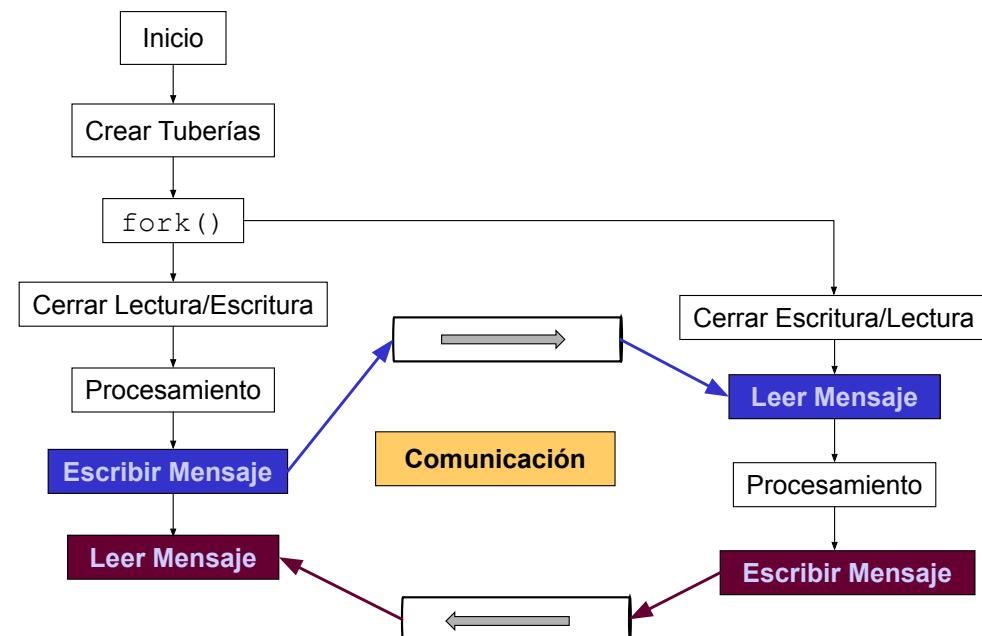
```
$ ls -l | wc -l
```



## Tuberías sin Nombre: Unidireccional



## Tuberías sin Nombre: Bidireccional



## Tuberías sin Nombre

- Crear una tubería:

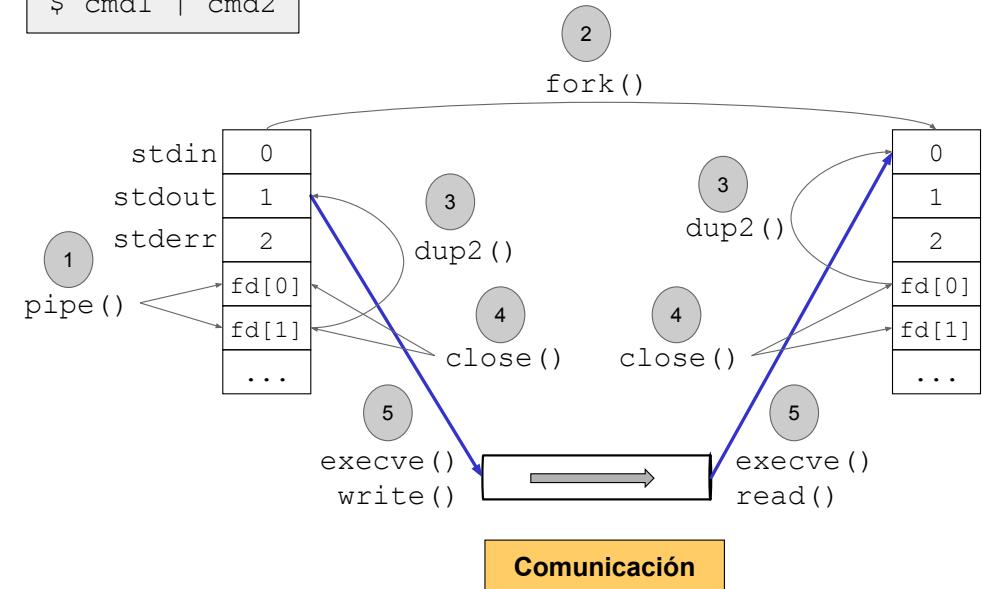
```
int pipe(int fd[2]);
```

<unistd.h>  
SV+BSD+POSIX

- Extremo de escritura: fd[1] Extremo de lectura: fd[0]
- Si la tubería está vacía, `read(2)` se bloqueará hasta que haya datos disponibles
  - Si la tubería se llena, `write(2)` se bloqueará hasta que se lean suficientes datos para que se pueda completar la escritura
  - Si todos los **descriptores de escritura se han cerrado**, `read(2)` devolverá cero, indicando el fin de fichero
  - Si todos los **descriptores de lectura se han cerrado**, `write(2)` enviará la señal `SIGPIPE` al proceso y, si se ignora la señal, fallará con `EPIPE`
  - Los descriptores que no sean necesarios deben cerrarse para asegurarse de que se notifica el fin de fichero o `SIGPIPE/EPIPE` cuando sea necesario

## Tuberías sin Nombre: Ejemplo

\$ cmd1 | cmd2



## Tuberías con Nombre

- La comunicación mediante **tuberías sin nombre** se puede realizar únicamente entre **procesos con relación de parentesco**
- Una **tubería con nombre**, o **fichero especial FIFO**, es similar a una tubería sin nombre, salvo que se accede como parte del sistema de ficheros
  - La entrada en el sistema de ficheros solo sirve para que los procesos abran la misma tubería con `open(2)` usando un nombre
  - El núcleo realiza la sincronización y almacena los datos internamente, sin escribirlos en el sistema de ficheros
  - El extremo de lectura se abre con `O_RDONLY` y el de escritura, con `O_WRONLY`
  - Varios procesos pueden abrir la tubería para lectura o escritura
- Deben abrirse ambos extremos antes de poder intercambiar datos
  - Normalmente, la apertura de un extremo se bloquea hasta que se abre el otro extremo
  - En modo no bloqueante (`flag O_NONBLOCK`), la apertura para lectura no se bloqueará aunque la tubería no esté abierta para escritura

## Tuberías con Nombre

- Crear ficheros especiales:

```
int mknod(const char *filename,  
          mode_t mode, dev_t dev);
```

- `filename` es el nombre del fichero (fichero, dispositivo, tubería) que se creará
- `mode` especifica los permisos (modificados por `umask`) y el tipo de fichero que se creará como OR lógica. El tipo ha de ser:

- `S_IFREG`: Fichero regular
- `S_IFCHR`: Dispositivo de caracteres (`dev = major,minor`)
- `S_IFBLK`: Dispositivo de bloques (`dev = major,minor`)
- `S_IFIFO`: Tubería con nombre
- `S_IFSOCK`: Socket UNIX

- El comando `mknod` proporciona acceso a esta funcionalidad:

```
mknod [-m permisos] nombre tipo
```

- `tipo` puede ser
  - `b`: Dispositivo de bloques
  - `c`: Dispositivo de caracteres
  - `p`: FIFO

<sys/types.h>  
<sys/stat.h>  
<fcntl.h>  
<unistd.h>

SV+BSD

## Tuberías con Nombre

- Crear tuberías con nombre:

```
int mkfifo(const char *filename,  
          mode_t mode);
```

- `filename` es el nombre de la tubería que se creará
- `mode` son los permisos (modificados por `umask`) con que se creará la tubería

- El comando `mkfifo` proporciona acceso a esta funcionalidad:

```
mkfifo [-m permisos] nombre
```



## Sincronización de E/S

- Cuando un proceso gestiona varios canales de E/S (tubería, socket o terminal), no debe bloquearse indefinidamente en uno de ellos mientras otros están listos para realizar una operación
- Alternativas:
  - E/S no bloqueante: Opción `O_NONBLOCK`
    - En lugar de bloquearse, la llamada falla con `errno=EAGAIN`
    - Es como la E/S por encuesta y consume tiempo de CPU innecesariamente, ya que el proceso nunca se bloquea, incluso si ningún descriptor está listo
  - E/S guiada por eventos: Opción `O_ASYNC`
    - El proceso recibe una señal (por defecto, `SIGIO`) cuando el descriptor está preparado para realizar la operación
    - La gestión de señales asíncronas modifica la lógica del programa
  - **Multiplexación de E/S síncrona:** Funciones `select()`, `poll()` y `epoll()`
    - El proceso monitoriza varios descriptores a la vez, esperando a que uno o varios estén listos para realizar una operación de E/S determinada de forma **síncrona**

## Multiplexación de E/S Síncrona

- Seleccionar descriptores de fichero preparados:

```
int select(int nfds, fd_set *readfds, fd_set *writefds,  
          fd_set *exceptfds, struct timeval *timeout);
```

- `readfds` es el conjunto de descriptores de lectura
- `writefds` es el conjunto de descriptores de escritura
- `exceptfds` es el conjunto de descriptores de condiciones excepcionales
  - Por ejemplo, que haya datos urgentes (*out-of-band*) en un socket TCP
- `nfds` es el mayor de los descriptores en los tres conjuntos, más 1
- `timeout` es el tiempo máximo en el que retornará la función
  - Si es `{0, 0}`, retorna inmediatamente
  - Si es `NULL`, se bloquea hasta que se produce un cambio
- Devuelve el número de descriptores listos o 0 si expira el tiempo máximo
  - Los conjuntos se modifican para indicar qué descriptores están listos para cada operación
- Si se produce un error, los conjuntos no se modifican y `timeout` queda indeterminado



## Multiplexación de E/S Síncrona

- Macros para la manipular los conjuntos:

```
void FD_ZERO(fd_set *set);  
void FD_SET(int fd, fd_set *set);  
void FD_CLR(int fd, fd_set *set);  
int FD_ISSET(int fd, fd_set *set);
```

- `FD_ZERO` inicializa un conjunto como conjunto vacío
- `FD_SET` añade un descriptor a un conjunto
- `FD_CLR` elimina un descriptor de un conjunto
- `FD_ISSET` comprueba si un descriptor está en un conjunto, lo cual es útil después de que `select()` retorne

```
...
fd_set rfds;
FD_ZERO(&rfds);
FD_SET(0, &rfds);

timeout.tv_sec = 2;
timeout.tv_usec = 0;

cambios = select(1, &rfds, NULL, NULL, &timeout);

if (cambios == -1)
    perror("select()");
else if (cambios) {
    read(0, buffer, 80);
    printf("Datos nuevos: %s\n", buffer);
} else {
    printf("Ningún dato nuevo en 2 seg.\n");
}
...
```

¿En qué política de planificación se usa el valor de *nice*?

- SCHED\_NICE.
- SCHED\_OTHER.
- SCHED\_RR.

¿En qué se diferencia una tubería sin nombre y una tubería con nombre?

- En la forma de crearse.
- En el patrón de comunicación.
- En que la comunicación en una se realiza en memoria y en otra, a través del sistema de ficheros.

¿Para qué sirve la opción SA\_RESTART al instalar un manejador de señal?

- Para reiniciar el manejador de señal por defecto tras tratar la señal.
- Para reiniciar la señal una vez tratada.
- Para reiniciar la llamada al sistema interrumpida.



# Introducción

## Sockets

- Cada extremo del canal de comunicación establecido entre el cliente y el servidor se denomina “socket” (enchufe)
- El socket permite el intercambio de datos bidireccional entre cliente y servidor
- Cada aplicación servidor o cliente está identificada (normalmente) por un número de puerto

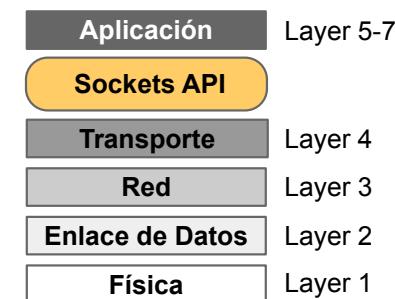


## TEMA 2.4. Programación con Sockets

### PROFESORES:

Rubén Santiago Montero  
Eduardo Huedo Cuesta  
Luis M. Costero Valero

- APIs de programación con sockets:
  - **BSD Sockets API** o Sockets de Berkeley
  - WinSock API, equivalente al de BSD
  - Bindings disponibles en todos los lenguajes



## Tipos de Sockets

- Especifican la semántica de la comunicación:
  - **SOCK\_STREAM**
    - Flujo de bytes orientado a conexión, con entrega ordenada, fiable y bidireccional
    - Debe establecerse la conexión para poder enviar o recibir datos
    - Similar a una tubería, se envía la señal SIGPIPE si un proceso envía en un flujo interrumpido
    - Los límites de los mensajes en los datagramas entrantes no se conservan, por lo que debe marcarse el inicio y fin del mensaje (ej. \n, <HTML>...</HTML>, {"msg": {...}})
  - **SOCK\_DGRAM**
    - Datagramas (mensajes de longitud máxima fija sin conexión y no fiables)
  - **SOCK\_RAW**
    - Acceso directo a los protocolos de red o de transporte, evitando el procesado normal de TCP/IP, lo que permite implementar nuevos protocolos en el espacio de usuario
    - Orientado a datagrama

## Protocolos de Soporte

- La abstracción ofrecida por los diferentes tipos de sockets se apoyan en las diferentes familias y protocolos de red
- Cada **tipo** de socket se implementa usando la funcionalidad de un **dominio** de comunicación
  - Algunos tipos de sockets pueden no estar soportados por todos los dominios
- Un **dominio** de comunicación es una familia de **protocolos** usados para comunicación que comparten un esquema de direccionamiento
  - **AF\_INET, AF\_INET6**: Protocolos de Internet sobre IPv4 e IPv6
  - **AF\_UNIX**: Comunicación local entre procesos de un mismo sistema
  - Otros: AF\_IPX, AF\_X25, AF\_APPLETALK, AF\_PACKET
- Se usa un **protocolo** particular de la familia para implementar cada **tipo** de socket
  - Normalmente, el tipo de socket determina el protocolo dentro de un dominio

# Direcciones de Sockets

- Direcciones de sockets IPv4:

```
struct sockaddr_in {  
    sa_family_t    sin_family; // Familia: AF_INET  
    in_port_t      sin_port;  // Puerto  
    struct in_addr sin_addr; // Dirección IPv4  
};  
  
struct in_addr {  
    uint32_t       s_addr;    // 32 bits dirección IP  
};
```

- **Puerto (in\_port\_t)**
  - Privilegiados (*well-known*) < 1024, sólo para procesos con privilegio
  - Asociados a los protocolos superiores TCP y UDP
- **Dirección (struct in\_addr)**
  - Dirección de red local o remota
  - Se puede inicializar o asignar con las constantes INADDR\_ANY (0.0.0.0) e INADDR\_LOOPBACK (127.0.0.1)

# Direcciones de Sockets

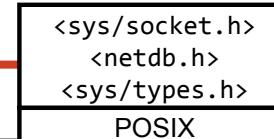
- Direcciones de sockets IPv6:

```
struct sockaddr_in6 {  
    sa_family_t    sin6_family; // Familia: AF_INET6  
    in_port_t      sin6_port;  // Número de puerto  
    uint32_t       sin6_flowinfo; // Id del flujo  
    struct in6_addr sin6_addr; // Dirección IPv6  
    uint32_t       sin6_scope_id; // Índ. de zona (link-local)  
};  
  
struct in6_addr {  
    unsigned char   s6_addr[16]; // Dir. IPv6 de 128 bits  
};  
  
○ La dirección IPv6 (struct in6_addr) se puede inicializar con las constantes IN6ADDR_ANY_INIT e IN6ADDR_LOOPBACK_INIT, o asignar a las variables in6addr_any (::) e in6addr_loopback (::1)
```

## Gestión de Direcciones

- Traducción de nombres a direcciones:

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



- node hace referencia al host, y puede ser:
  - Un nombre de host, que se resuelve usando gethostbyname(3)
  - Una dirección IPv4 en notación decimal de punto (ej. "192.168.0.1")
  - Una dirección IPv6 en notación hexadecimal abreviada (ej. "fe80::1:2")
  - NULL, para especificar el host local
- service hace referencia al puerto, y puede ser:
  - Un nombre del servicio, según /etc/services (ej. "http")
  - Un número entero en decimal (ej. "80")
  - NULL, para no especificar ninguno
- hints establece algunos criterios de búsqueda
- res se usa para devolver una lista de direcciones de socket
  - El host tiene varios interfaces o soporta varios protocolos (ej. IPv4 e IPv6)
  - El servicio soporta varios protocolos (ej. telnet → tcp/23 y udp/23)

## Gestión de Direcciones

```
struct addrinfo {  
    int           ai_flags; // Opciones para filtrado (hints)  
    int           ai_family;  
    int           ai_socktype;  
    int           ai_protocol;  
    socklen_t     ai_addrlen; // Resultado (res)  
    struct sockaddr *ai_addr;  
    char          *ai_canonname;  
    struct addrinfo *ai_next;  
};
```

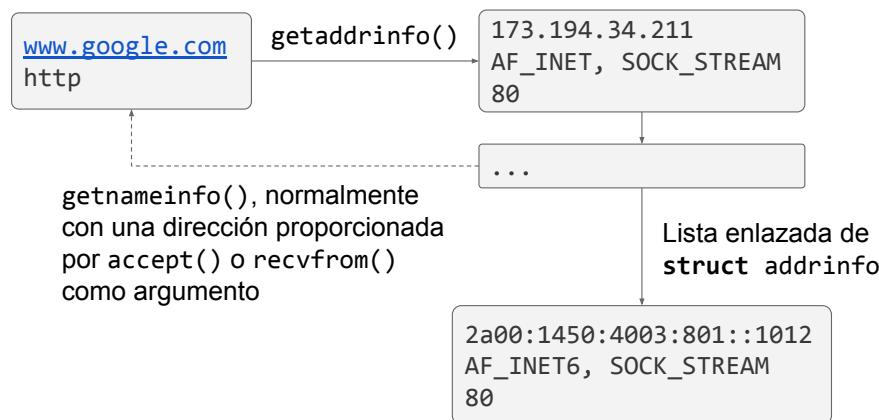
- Opciones de filtrado (hints, el resto de campos deben ser 0 o NULL):
  - ai\_family: AF\_INET para IPv4, AF\_INET6 para IPv6 o AF\_UNSPEC para ambos
  - ai\_socktype y ai\_protocol: Tipo de socket y protocolo
  - ai\_flags: Opciones, ej. AI\_PASSIVE para devolver 0.0.0.0 ó :: si node=NULL (si no, devuelve 127.0.0.1 ó ::1)
- Resultado (res):
  - ai\_addr y ai\_addrlen: Puntero a la dirección y tamaño en bytes
  - ai\_canonname: Nombre oficial del host si AI\_CANONNAME en ai\_flags
  - ai\_next: Puntero al siguiente resultado (lista enlazada)

## Gestión de Direcciones

- Traducción de direcciones a nombres:

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

<sys/socket.h>  
<netdb.h>  
<sys/types.h>  
POSIX



## Conversión de Direcciones y Valores

- Convertir direcciones entre formato binario y 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);
```

- af es una familia de protocolos (AF\_INET o AF\_INET6)
- Los argumentos de tipo `void *` contienen la estructura de la dirección en binario (`struct in_addr` o `struct in6_addr`)
- Los argumentos de tipo `char *` contienen la representación de la dirección como texto (representación decimal de punto o hexadecimal abreviada)

- Convertir valores entre orden de byte de red y de host:

```
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

- Los datos se envían en orden de byte de red (*big-endian*), por lo que puede ser necesario convertirlos al orden de la arquitectura del procesador
- Las direcciones y puertos se almacenan en orden de byte de red

## Creación de Sockets

- Crear un socket:

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

<sys/types.h>  
<sys/socket.h>

POSIX+BSD

- domain es la familia de protocolos
- type es el tipo de socket
- protocol puede ser:
  - IPPROTO\_TCP para SOCK\_STREAM ⇒ Usar siempre 0
  - IPPROTO\_UDP para SOCK\_DGRAM ⇒ Usar siempre 0

- Devuelve un descriptor de fichero para el socket

- Creación de sockets IPv4:

```
tcp_sd = socket(AF_INET, SOCK_STREAM, 0);
udp_sd = socket(AF_INET, SOCK_DGRAM, 0);
```

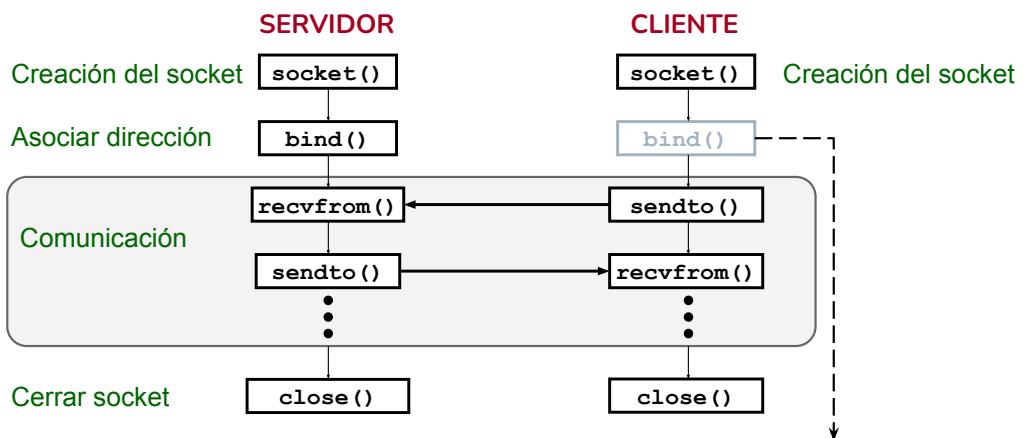
- Creación de sockets IPv6:

```
tcp6_sd = socket(AF_INET6, SOCK_STREAM, 0);
udp6_sd = socket(AF_INET6, SOCK_DGRAM, 0);
```

- La implementación de IPv6 es compatible casi totalmente con IPv4

## Sockets UDP: Patrón de Comunicación

- Sockets tipo SOCK\_DGRAM para la familia AF\_INET y AF\_INET6

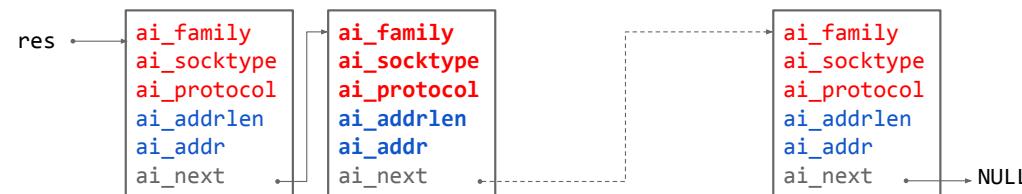


Asocia el socket a una dirección local.  
Si no, elige INADDR\_ANY y un puerto libre aleatorio (puerto efímero)

NOTA: Este patrón de comunicaciones es también válido para AF\_UNIX

# Asignación de Direcciones

```
getaddrinfo(node, service, &hints, &res);
```



```
sock = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
// Servidor
bind(sock, (struct sockaddr *) res->ai_addr, res->ai_addrlen);
// Cliente
connect(sock, (struct sockaddr *) res->ai_addr, res->ai_addrlen);
```

# Envío y Recepción de Datos

POSIX

<sys/socket.h>

- Enviar y recibir datos:

```
ssize_t sendto(int sd, const void *buf, size_t len, int flags,
               const struct sockaddr *dst_addr, socklen_t addrlen);
ssize_t recvfrom(int sd, void *buf, size_t len, int flags,
                 struct sockaddr *src_addr, socklen_t *addrlen);
```

- Con SOCK\_DGRAM se usan para poder especificar u obtener la dirección del otro extremo (con SOCK\_STREAM se ignora la dirección)
- recvfrom(2) se bloquea si no hay mensajes disponibles:
  - Se puede usar select() o el modo no bloqueante (flag MSG\_DONTWAIT)

# Almacenamiento de Direcciones

- Para escribir aplicaciones compatibles con IPv4 e IPv6, deben eliminarse las dependencias en el formato de las direcciones
  - struct sockaddr solo evita advertencias del compilador (tiene el mismo tamaño que struct sockaddr\_in)
  - La nueva struct sockaddr\_storage permite almacenar tanto struct sockaddr\_in como struct sockaddr\_in6
- Ejemplo:

```
struct sockaddr_storage addr;
socklen_t addrlen = sizeof(addr);
b = recvfrom(sd, buf, 80, 0, (struct sockaddr *) &addr, &addrlen);
```

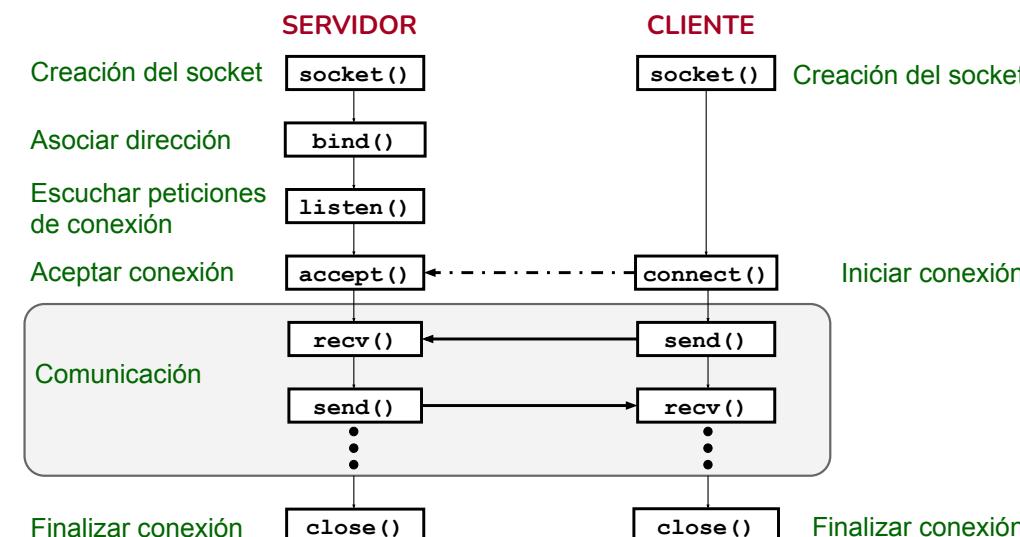
- addr se usa para devolver una dirección IPv4 o IPv6
- addrlen es un argumento valor-resultado, que inicialmente contiene el tamaño del búfer al que apunta addr, y se modifica para indicar el tamaño real de la dirección devuelta

# Resumen: Esquema Servidor UDP

```
hints.ai_flags      = 0;
hints.ai_family    = AF_UNSPEC; // IPv4 o IPv6
hints.ai_socktype  = SOCK_DGRAM;
rc = getaddrinfo(argv[1], argv[2], &hints, &result);
sd = socket(result->ai_family, result->ai_socktype, 0);
bind(sd, (struct sockaddr *) result->ai_addr, result->ai_addrlen);
while (1) {
    addrlen = sizeof(addr);
    c = recvfrom(sd, buf, 80, 0, (struct sockaddr *) &addr, &addrlen);
    getnameinfo((struct sockaddr *) &addr, addrlen, host, NI_MAXHOST,
                serv, NI_MAXSERV, NI_NUMERICHOST|NI_NUMERICSERV);
    printf("Recibidos %d bytes de %s:%s\n", c, host, serv);
    sendto(sd, buf, c, 0, (struct sockaddr *) &addr, addrlen);
}
```

# Sockets TCP: Patrón de Comunicación

- Sockets de tipo SOCK\_STREAM para la familia AF\_INET y AF\_INET6



NOTA: Este patrón de comunicaciones es también válido para AF\_UNIX

## Envío y Recepción de Datos

- Enviar y recibir datos:

```
ssize_t send(int sd, const void *buff, size_t len, int flags);
ssize_t recv(int sd, void *buffer, size_t len, int flags);
```

- Se usan normalmente con sockets SOCK\_STREAM
- send(2) envía len bytes de buffer
  - Con SOCK\_DGRAM hay que usar connect(2) previamente
  - Si el mensaje es demasiado grande, no se envían datos (EMSGSIZE)
- recv(2) recibe hasta len bytes en buffer
  - Con SOCK\_DGRAM, el mensaje se debe leer en una sola operación (tamaño del buffer) para no perder datos
- Ambas pueden bloquearse
  - send(2) se bloquea si el mensaje no cabe en el buffer de envío
  - recv(2) se bloquea si no hay mensajes disponibles en el socket
  - Se puede usar select() o el modo no bloqueante (flag MSG\_DONTWAIT)

## Gestión de la Conexión

POSIX

<sys/socket.h>

- Escuchar conexiones en un socket:

```
int listen(int sd, int backlog);
```

- Con SOCK\_STREAM se usa en el servidor para poner el socket en modo de escucha para aceptar peticiones de conexión
- backlog es el tamaño máximo de la cola de conexiones completamente establecidas esperando ser aceptadas
  - No confundir con la cola SYN de conexiones incompletas

- Aceptar una conexión en un socket:

```
int accept(int sd, struct sockaddr *addr, socklen_t *addrlen);
```

- Con SOCK\_STREAM se usa en el servidor para aceptar una conexión establecida y crear un nuevo socket conectado
- Se bloquea si no hay conexiones pendientes
  - Se puede usar select() o el modo no bloqueante
- addr devuelve la dirección del cliente que se ha conectado
- addrlen indica el tamaño del búfer y devuelve el tamaño real de la dirección
- Devuelve un descriptor de socket para gestionar la conexión

## Resumen: Esquema Servidor TCP

POSIX  
<sys/socket.h>

```
hints.ai_flags      = 0;
hints.ai_family     = AF_UNSPEC;      // IPv4 o IPv6
hints.ai_socktype   = SOCK_STREAM;
rc = getaddrinfo(argv[1], argv[2], &hints, &result);
sd = socket(result->ai_family, result->ai_socktype, 0);
bind(sd, (struct sockaddr *) result->ai_addr, result->ai_addrlen);
listen(sd, 5);

while (1) {
    addrlen = sizeof(addr);
    clisd = accept(sd, (struct sockaddr *) &addr, &addrlen);
    getnameinfo((struct sockaddr *) &addr, addrlen, host, NI_MAXHOST,
                serv, NI_MAXSERV, NI_NUMERICHOST|NI_NUMERICSERV);
    printf("Conexión desde %s:%s\n", host, serv);
    while (c = recv(clisd, buf, 80, 0)) { // Comprobar mensaje!
        send(clisd, buf, c, 0);
    }
    close(clisd);
}
```

# Opciones de Sockets

POSIX
<sys/socket.h>

- Pueden fijarse y consultarse diversas opciones en el socket:

```
int setsockopt(int sd, int level, int optname,
               const void *optval, socklen_t optlen);
int getsockopt(int sd, int level, int optname,
               void *optval, socklen_t *optlen);
```

- level especifica el nivel de la capa de protocolos donde aplica la opción:
  - API de sockets: SOL\_SOCKET
  - Protocolo: IPPROTO\_IP, IPPROTO\_IPV6, IPPROTO\_TCP, IPPROTO\_UDP
- optname especifica la opción, que acepta un valor optval de un tipo específico (**void \***) y tamaño optlen
- Muchas opciones se pueden configurar vía sysctl o /proc

# Opciones para Sockets

- Nivel de API de sockets (SOL\_SOCKET):
  - SO\_KEEPALIVE: Activa el mecanismo de *keepalive* en sockets SOCK\_STREAM
  - SO\_BROADCAST: Permite a sockets SOCK\_DGRAM usar direcciones de *broadcast*
  - SO\_REUSEADDR: Activa la reutilización de direcciones locales en TIME\_WAIT
  - SO\_SNDBUF y SO\_RCVBUF: Obtienen o establecen el tamaño de los buffers de envío y recepción (actualmente se autoajusta en función de la latencia y el ancho de banda)
- Nivel de protocolo TCP (IPPROTO\_TCP):
  - TCP\_NODELAY: Desactiva el algoritmo de Nagle
  - TCP\_QUICKACK: Desactiva los ACKs retrasados
- Nivel de protocolo IPv4 (IPPROTO\_IP):
  - IP\_ADD\_MEMBERSHIP e IP\_DROP\_MEMBERSHIP: Gestión de grupos multicast
  - IP\_MTU: Obtiene el MTU de la ruta
  - IP\_MTU\_DISCOVER: Activa el algoritmo Path MTU Discovery
  - IP\_OPTIONS, IP\_TTL e IP\_TOS: Obtienen o establecen campos del datagrama

# Soporte para Clientes IPv4 e IPv6

- Alternativas para servidores que soporten clientes IPv4 e IPv6
- Crear un único socket IPv6 para ambas versiones (*dual stack*):
  - Deshabilitar la opción IPV6\_V6ONLY en el socket (su valor se define en el parámetro net.ipv6.bindv6only, que por defecto está deshabilitado)

```
int on = 0;
setsockopt(sd, IPPROTO_IPV6, IPV6_V6ONLY, (void *) &on,
            sizeof(on));
```
  - Asociar (con bind()) a :: (in6addr\_any)
    - Usa direcciones IPv6 mapeadas a IPv4 (192.168.0.1 ⇒ ::FFFF:192.168.0.1)
    - No está soportado en todos los sistemas
- Crear dos sockets, uno para cada versión:
  - Habilitar IPV6\_V6ONLY en el socket IPv6 si se va a asociar a ::
  - Obtener las direcciones válidas para crear un socket con cada versión

# Servidores Concurrentes

## Necesidad

- El servidor debe atender a varios clientes concurrentemente
- En general, las llamadas son bloqueantes
  - accept(2) espera a que se establezcan conexiones de clientes
  - recv(2) y recvfrom(2) esperan a que lleguen de datos
  - send(2) espera si el mensaje no cabe en el *buffer* de envío

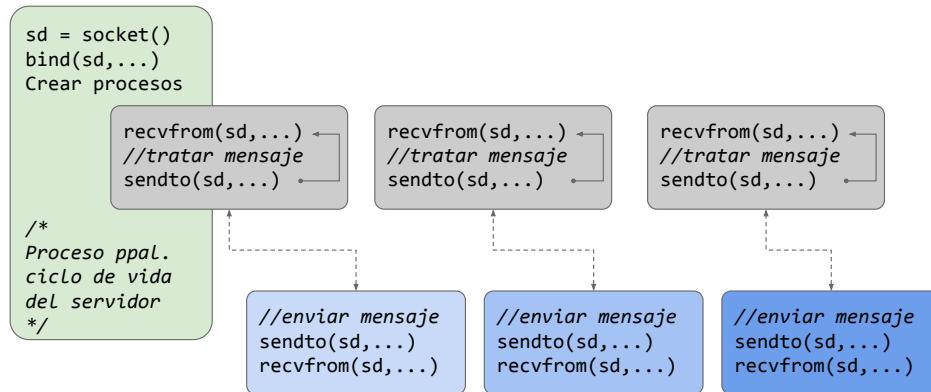
## Herramientas

- Los threads comparten un espacio de direcciones y los descriptores (sockets) y los procesos heredan los descriptores (sockets)
- Las operaciones son concurrentes sobre descriptores de socket
  - Múltiples threads pueden llamar a accept() para establecer una conexión
  - Múltiples threads pueden llamar a recvfrom() para recibir datos
  - Todos los threads se bloquearán en la llamada y solo uno de ellos se desbloqueará cuando llegue una solicitud de conexión o datos
- También se puede usar multiplexación de E/S síncrona (select()), pero la lógica del programa es mucho más compleja

# Servidores Concurrentes: SOCK\_DGRAM

## Patrón

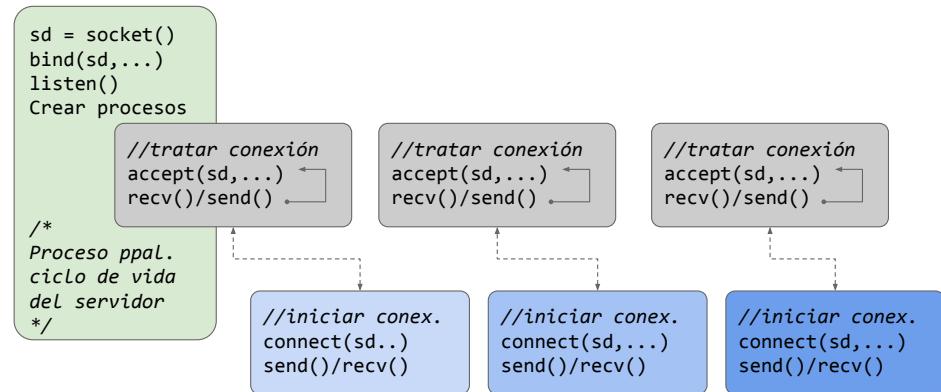
- Recepción concurrente de mensajes
- El servidor crea un conjunto de procesos/threads para procesar los mensajes recibidos con `recvfrom()`
- La concurrencia es en el nivel del mensaje



# Servidores Concurrentes: SOCK\_STREAM

## Patrón pre-fork

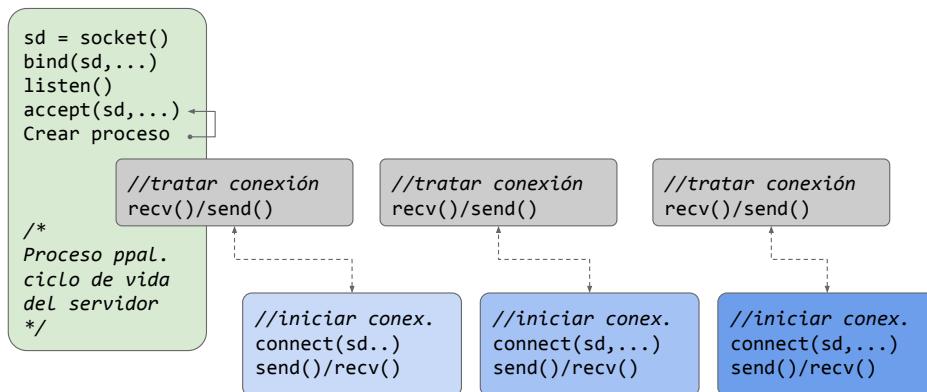
- Gestión concurrente de conexiones
- El servidor crea un conjunto de procesos/threads que aceptan conexiones con `accept()` y las gestionan
- La concurrencia es en el nivel de conexión



# Servidores Concurrentes: SOCK\_STREAM

## Patrón accept-and-fork

- Gestión concurrente de conexiones
- El servidor acepta conexiones con `accept()` y crea un proceso/thread para procesar cada una
- La concurrencia es en el nivel de conexión



# Ejemplos de Preguntas Teóricas

¿Generan algún mensaje de red `socket(2)`, `bind(2)`, `listen(2)` y `accept(2)`?

- No.  
 Si.  
 Depende.

¿Qué ocurre si no se usa la llamada `bind(2)` en un socket cliente?

- Que la siguiente llamada dará error.  
 El resultado es indefinido, porque es obligatorio usarla.  
 Que se asocia a la dirección local `INADDR_ANY` y a un puerto libre aleatorio.

Umask (user mask o máscara de usuario), es un comando para el entorno POSIX que establece las autorizaciones predeterminadas que se fijan cuando se crea un archivo, directorio o carpeta.

Esta también le hace referencia a la función que ejerce la máscara (mask), y a la máscara en sí; comúnmente se le conoce como la máscara de creación del modo de archivo.

Valores predeterminados de umask:

Los valores predeterminados pueden cambiar dependiendo del administrador de sistemas que lo manifieste, ya que cada uno posee sus prioridades, o solamente están acostumbrados a definir sus propios valores dependiendo del sistema que este maneje.

Algo cierto es que el valor umask en linux se determina según la finalidad de estos, como archivos, directorios, directorio de inicio para cualquier usuario, entre otros. En la siguiente tabla vemos los de uso general (común).

#### VALOR MASK ARCHIVOS DIRECTORIOS

000 666 (rw-rw-rw-) 777 (rwxrwxrwx)

002 664 (rw-rw-r-) 775 (rwxrwxr-x)

007 660 (rw-rw--) 770 (rwxrwx--)

022 644 (rw-r-r-) 755 (rwxr-xr-x)

27 640 (rw-r--) 750 (rwxr-x--)

77 600 (rw----) 700 (rwx---)

277 400 (r----) 500 (r-x---)

Cómo calcular el valor umask:

En la siguiente tabla observaremos la equivalencia de la notación octal con los permisos linux.



Umask	Ficheros creados	Binario	Directorios creados	Binario
000	666 (rw-rw-rw-)	110110110	777 (rwxrwxrwx)	111111111
002	664 (rw-rw-r--)	110110100	775 (rwxrwxr-x)	111111101
022	644 (rw-r--r--)	110100100	755 (rwxr-xr-x)	111101101
027	640 (rw-r-----)	110100000	750 (rwxr--x---)	111101000
077	600 (rw-----)	110000000	700 (rwx-----)	111000000
277	400 (r-----)	100000000	500 (r-x-----)	101000000

666.

Según lo previsto hasta ahora, para dar las autorizaciones predeterminados 755 se produce lo siguiente:  $777 - 755 = 022$ . Lo que significa que el valor umask para obtener la autorización 755, es 022.

Pero si por ejemplo deseamos que la autorización predefinida sea 700, siendo así sería:  $777 - 700 = 077$  con un valor umask de 077

Valor octal Permisos linux

0 Sin permiso

1 Solo ejecutar

2 Solo escribir

3 Ejecutar y escribir

4 Solo lectura

5 Leer y ejecutar

6 Leer y escribir

7 Permiso total

Valor de umask:

Según hemos podido observar en la tabla anterior, el la autorización total para estos directorios o carpetas es 777, si aplicamos en los archivos la autorización máxima sería 666.