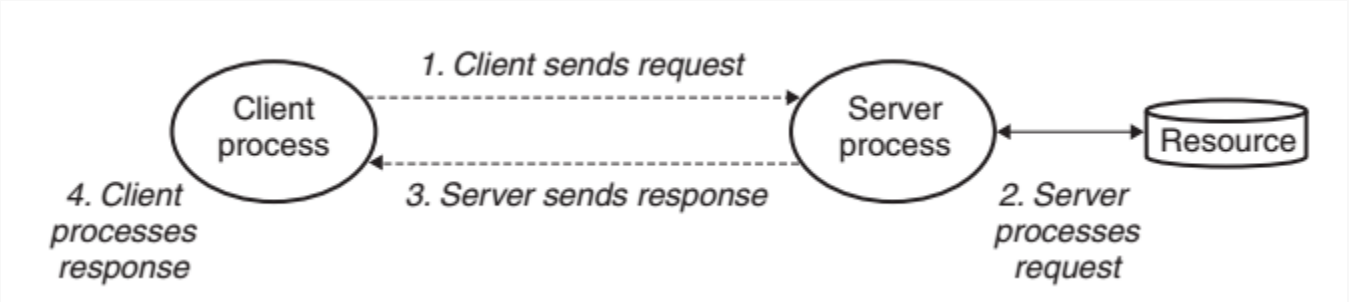
PROGRAMACIÓN DE SISTEMAS

Unidad 4 – Procesamiento Remoto

PROTOCOLOS DE COMUNICACIÓN REMOTA

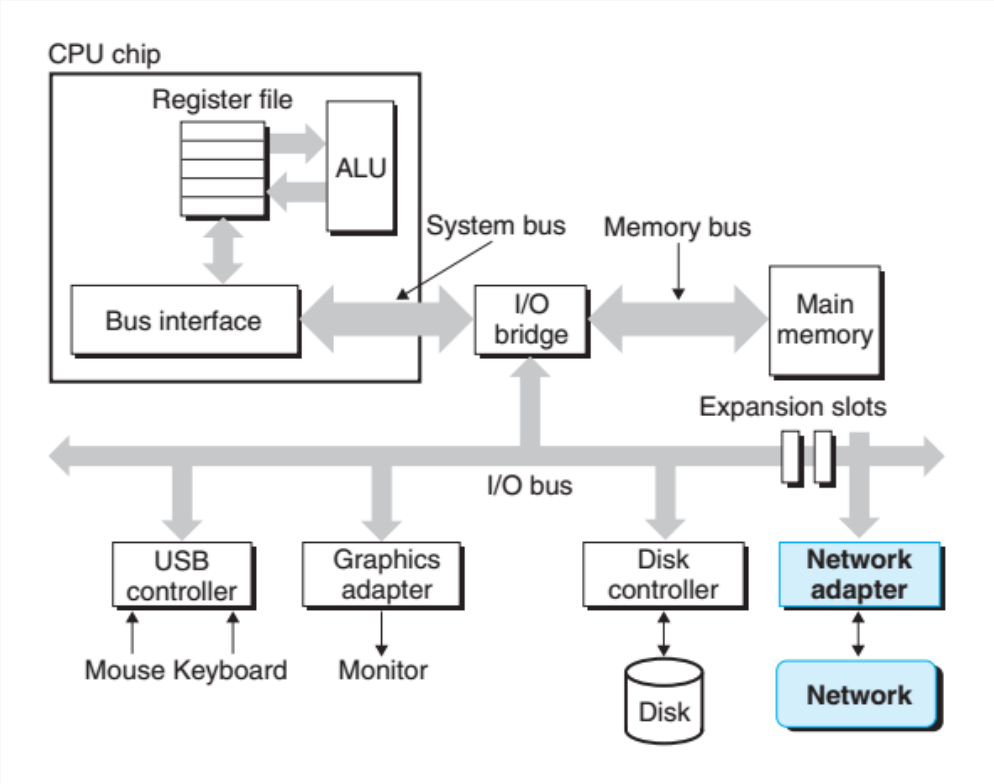
Protocolos de Comunicación

Modelo Cliente-Servidor



Protocolos de Comunicación

Redes



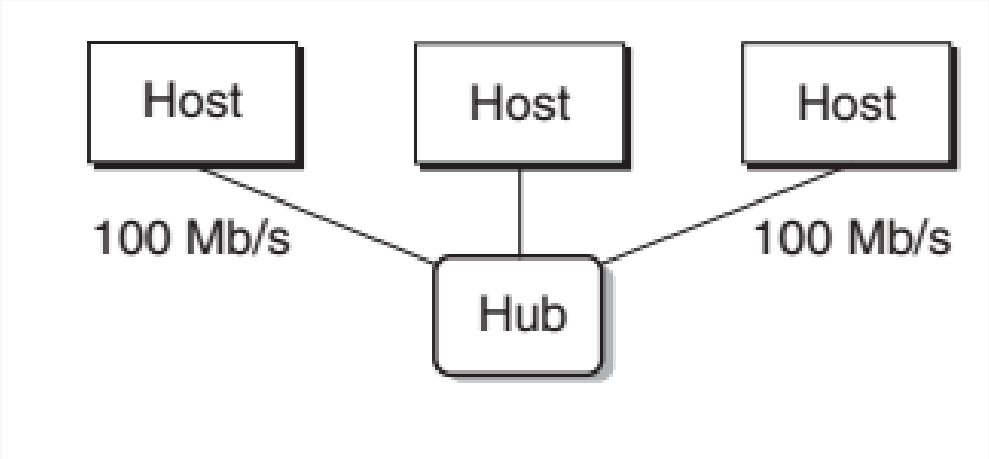
Protocolos de Comunicación

Redes

- Físicamente, red es un sistema jerárquico organizado por proximidad.
- Tecnólogia más común es Ethernet.

Protocolos de Comunicación

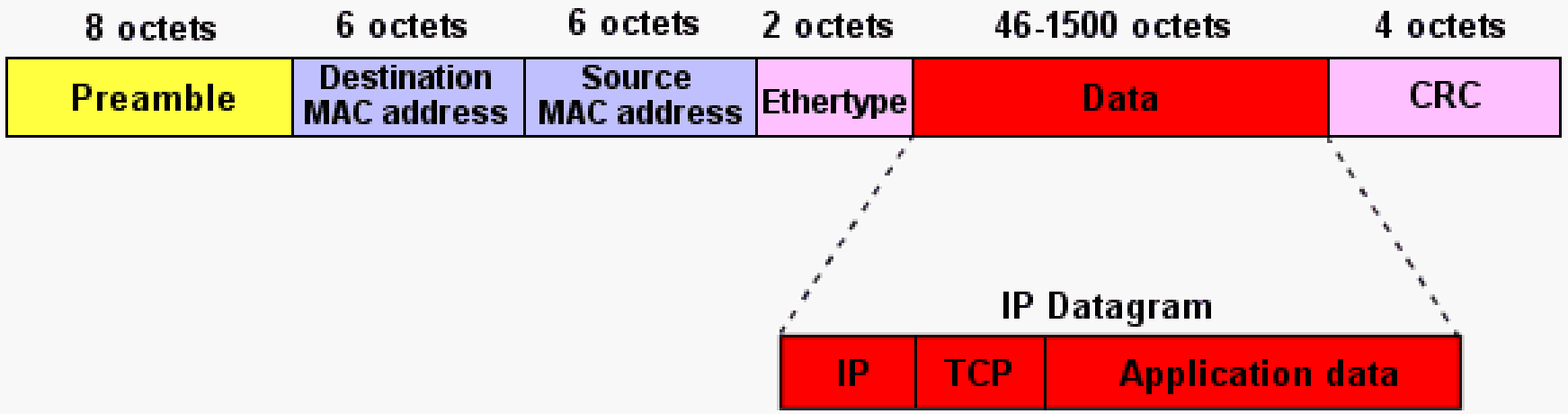
Segmentos de Ethernet



Protocolos de Comunicación

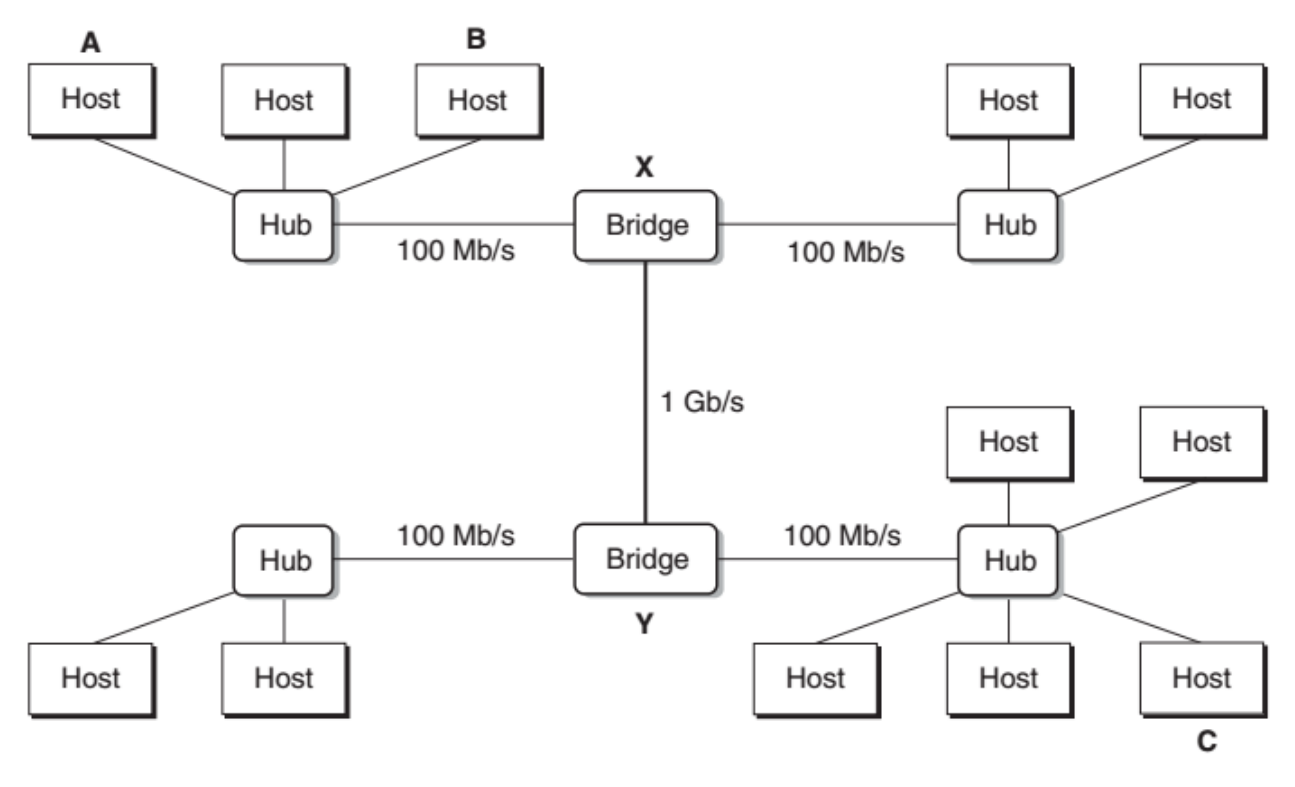
- Cada adaptador tiene direccion de 48-bits
- Host puede mandar pedazos de información, llamados **frames.**
- Cada frame tiene una **cabecera**
- Al enviar frame, solo host con dirección de destino procesa el frame.

Protocolos de Comunicación



Protocolos de Comunicación

Bridges



Protocolos de Comunicación

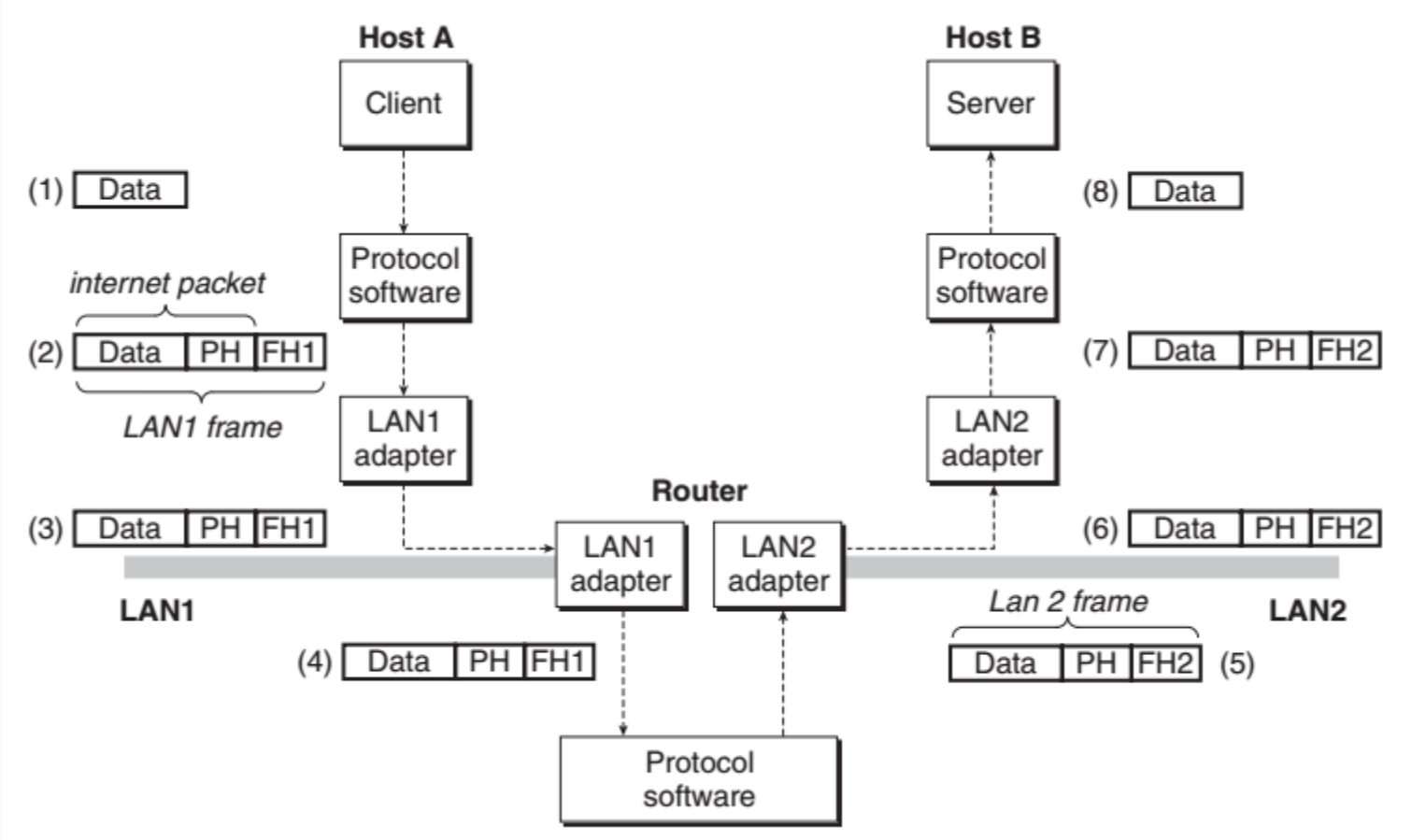
Protocolos

¿Cómo enviar información entre todas estas redes?

Software de protocolo (en hosts y routers). Capacidades:

- Mecanismo de nombres
- Mecanismos de entrega

Protocolos de Comunicación



Protocolos de Comunicación

Internet

Todo host conectado implementa el protocolo TCP/IP (?)

En Linux, podemos acceder a estos servicios a través de la **interfaz de sockets.**

Protocolos de Comunicación

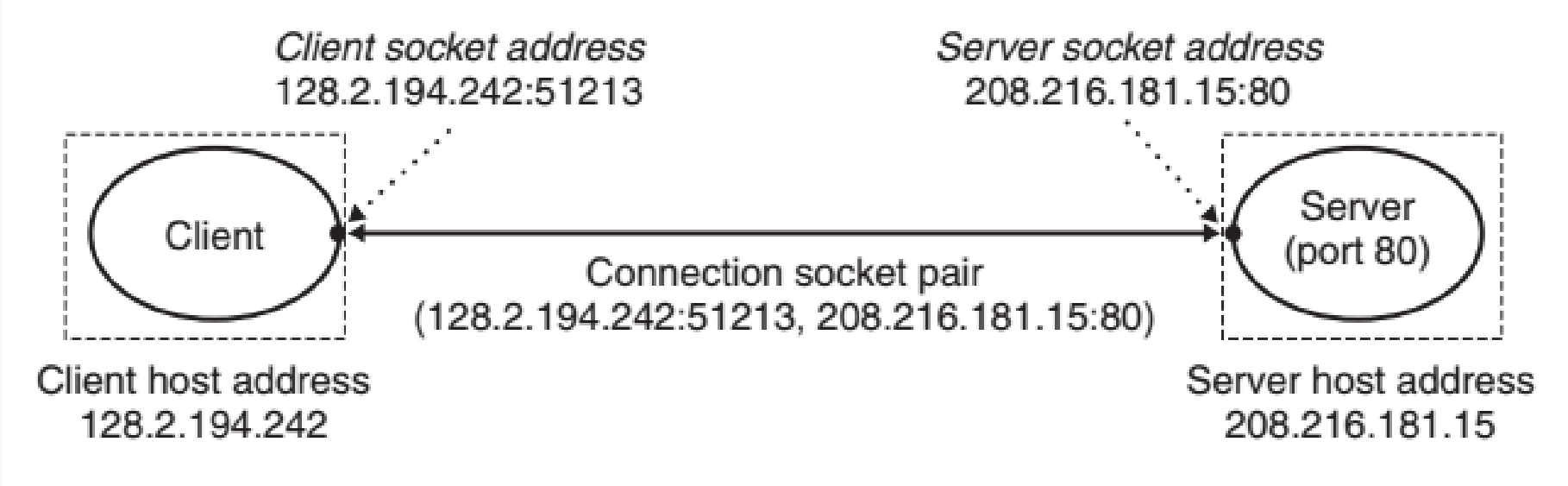
Internet

Para nosotros, Internet lo podemos concebir como colección de hosts con estas propiedades:

1. Cada host tiene dirección de 32 bits (dirección IP)
2. Direcciones están mapeadas a nombres de dominio
3. Proceso en un host se comunica con proceso de otro host usando una conexión

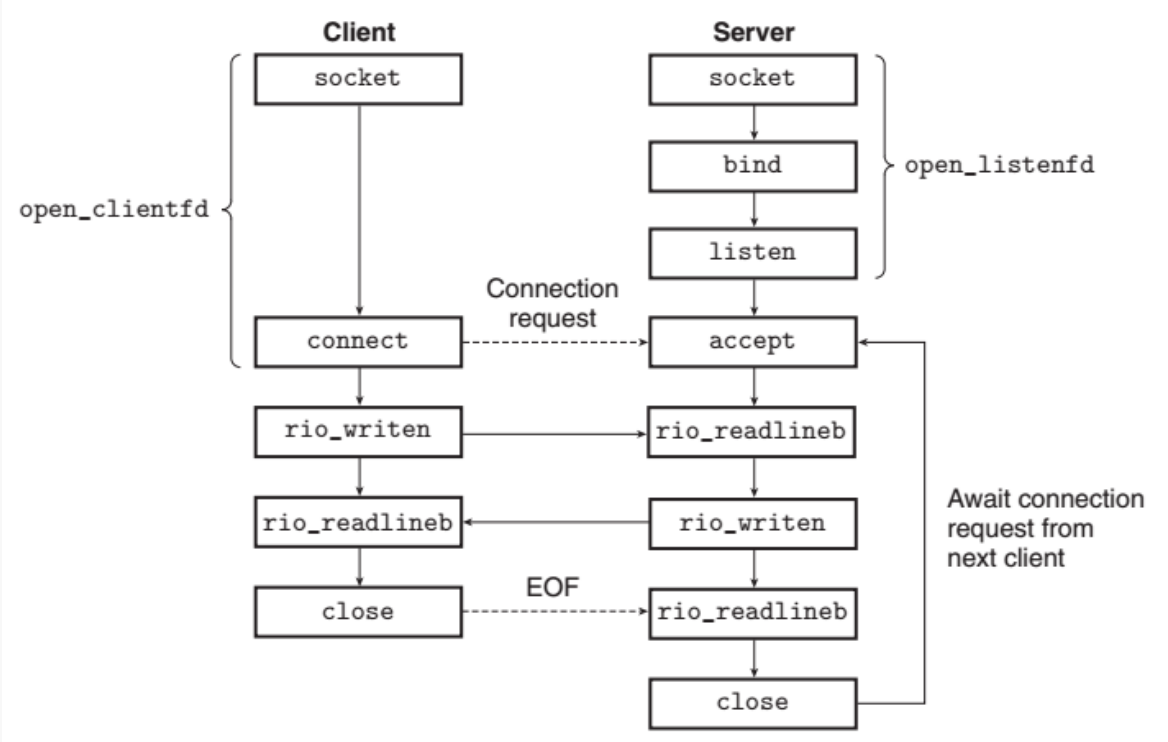
Protocolos de Comunicación

Conexiones



Protocolos de Comunicación

La interfaz de sockets



Protocolos de Comunicación

Llamada socket

- Los sockets de red de Linux, operan en la capa de red/transporte.

```
#include <sys/socket.h>

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

- Esta función nos devuelve un **descriptor de socket** (un numero entero)**.**

Protocolos de comunicación

Dominios:

Domain	Description
AF_INET	IPv4 Internet domain
AF_INET6	IPv6 Internet domain (optional in POSIX.1)
AF_UNIX	UNIX domain
AF_UNSPEC	unspecified

Tipos de socket:

Type	Description
SOCK_DGRAM	fixed-length, connectionless, unreliable messages
SOCK_RAW	datagram interface to IP (optional in POSIX.1)
SOCK_SEQPACKET	fixed-length, sequenced, reliable, connection-oriented messages
SOCK_STREAM	sequenced, reliable, bidirectional, connection-oriented byte streams

Protocolos de Comunicación

- SOCK_DGRAM: mensajes de longitud fija, sin conexión, poco confiables.
- SOCK_RAW: interface a IP.
- SOCK_SEQPACKET: longitud fija, con secuencia, confiables, orientados a conexión.
- SOCK_STREAM: stream de bytes secuenciales, confiables, bidireccionales, orientados a conexión.

Protocolos de Comunicación

- El argumento `protocol` define el protocolo de comunicación a usar.
- Usualmente es cero, en cuyo caso `type` y `domain` usarán el protocolo por defecto.
  - *AF_INET + SOCK_DGRAM = UDP*
  - *AF_INET + SOCK_STREAM = TCP*

Protocol	Description
IPPROTO_IP	IPv4 Internet Protocol
IPPROTO_IPV6	IPv6 Internet Protocol (optional in POSIX.1)
IPPROTO_ICMP	Internet Control Message Protocol
IPPROTO_RAW	Raw IP packets protocol (optional in POSIX.1)
IPPROTO_TCP	Transmission Control Protocol
IPPROTO_UDP	User Datagram Protocol

Protocolos de Comunicación

Direccionamiento

- Una vez creado el socket, debemos poder identificar el proceso a donde que queremos transmitir. Para esto usamos:
  1. ***Dirección de red*** *del equipo*
  2. ***Numero de puerto****: identifica el servicio con el cual nos contactamos*

Protocolos de Comunicación

- Orden de bytes de Red

```
#include <netinet/in.h>

unsigned long int htonl(unsigned long int hostlong);
unsigned short int htons(unsigned short int hostshort);
                                        Returns: value in network byte order

unsigned long int ntohl(unsigned long int netlong);
unsigned short int ntohs(unsigned short int netshort);
                                        Returns: value in host byte order
```

Protocolos de Comunicación

- Para convertir la dirección de bytes a formato a.b.c.d (ej.: 10.24.13.233), usamos:

```
#include <arpa/inet.h>

in_addr_t inet_addr(const char *cp);

char *inet_ntoa(struct in_addr *in);

int inet_pton( int domain, const char *restrict str,
                 void *restrict addr);

const char *inet_ntop( int domain, const void *restrict addr,
                 char *restrict str, socklen_t size);
```

Protocolos de comunicación

Formatos de dirección

```
/* para connect, bind y accept */
struct sockaddr {
        sa_family_t sa_family; /* address family */
        char sa_data[14];      /* variable-length address */
};


struct in_addr {
        in_addr_t s_addr; /* IPv4 address */
};

struct sockaddr_in {
        sa_family_t sin_family; /* address family */
        in_port_t sin_port; /* port number */
        struct in_addr sin_addr; /* IPv4 address */
};
```

Protocolos de Comunicación

Asociando direcciones con sockets

- Muy importante para servidores

```
#include <sys/socket.h>

int bind( int sockfd, const struct sockaddr *addr, socklen_t len);
```

- Si `addr` es INADDR_ANY, socket estará ligado a todas las interfaces de red de la máquina.

Protocolos de Comunicación

Restricciones:

1. Dirección debe ser válida para la máquina donde corre el proceso
2. La dirección debe ser del formato que especifica la familia de dirección que especificamos cuando creamos el socket.
3. Puerto debe ser mayor de 1024, si es que no somos root.
4. Usualmente, **un** solo socket puede ser ligado a una dirección.

Protocolos de Comunicación

Estableciendo una conexión

- Necesario para SOCK_STREAM o SOCK_SEQPACKET.

```
#include <sys/socket.h>

int connect(int sockfd, const struct sockaddr *addr, socklen_t len);
```

- addr es la dirección de equipo al que nos queremos comunicar.
- **¡Debemos poder manejar los errores que nos devuelva esta función!**
- **¡Estado del socket es INDEFINIDO si connect falla!**

Protocolos de Comunicación

Escuchando en el Servidor

Servidor dice que esta dispuesto a recibir conexiones usando:

```
#include <sys/socket.h>

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

- `backlog` indica cuando solicitudes pendientes podrá tener el socket antes de rechazarlas.

Protocolos de Comunicación

Aceptando Conexiones

```
#include <sys/socket.h>

int accept( int sockfd, struct sockaddr *restrict addr,
            socklen_t *restrict len);
```

- Nos devuelve un descriptor de socket asociado al equipo que llamó `connect`. El nuevo descriptor tiene las mismas características de sockfd.
- El socket `sockfd` que pasamos a `accept` se mantiene disponible para continuar recibiendo conexiones.
- `accept` bloquea al proceso hasta que llegue una conexión.

Protocolos de Comunicación

Transfiriendo información

- Podemos usar read(2) y write(2).
- Para enviar datos:

```
#include < sys/socket.h >

ssize_t send( int sockfd, const void *buf, size_t nbytes, int flags);
```

Flag	Description	POSIX.1	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10
MSG_CONFIRM	Provide feedback to the link layer to keep address mapping valid.			•		
MSG_DONTROUTE	Don't route packet outside of local network.		•	•	•	•
MSG_DONTWAIT	Enable nonblocking operation (equivalent to using O_NONBLOCK).		•	•	•	•
MSG_EOF	Shut the sender side of the socket down after sending data.		•		•	
MSG_EOR	Mark the end of the record if supported by protocol.	•	•	•	•	•
MSG_MORE	Delay sending the packet to allow more data to be written.			•		
MSG_NOSIGNAL	Don't generate SIGPIPE when writing to an unconnected socket.	•	•	•		
MSG_OOB	Send out-of-band data if supported by protocol (see Section 16.7).	•	•	•	•	•

Protocolos de Comunicación

- Para recibir información, usamos `recv`:

```
#include < sys/socket.h >

ssize_t recv( int sockfd, void *buf, size_t nbytes, int flags);
```

- Las banderas `flags` pueden ser las mostradas en la siguiente diapositiva.

Flag	Description	POSIX.1	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10
MSG_CMSG_CLOEXEC	Set the close-on-exec flag for file descriptors received over a UNIX domain socket (see Section 17.4).			•		
MSG_DONTWAIT	Enable nonblocking operation (equivalent to using O_NONBLOCK).		•	•		•
MSG_ERRQUEUE	Receive error information as ancillary data.			•		
MSG_OOB	Retrieve out-of-band data if supported by protocol (see Section 16.7).	•	•	•	•	•
MSG_PEEK	Return packet contents without consuming the packet.	•	•	•	•	•
MSG_TRUNC	Request that the real length of the packet be returned, even if it was truncated.			•		
MSG_WAITALL	Wait until all data is available (SOCK_STREAM only).	•	•	•	•	•