Clase de Sockets en lenguaje C

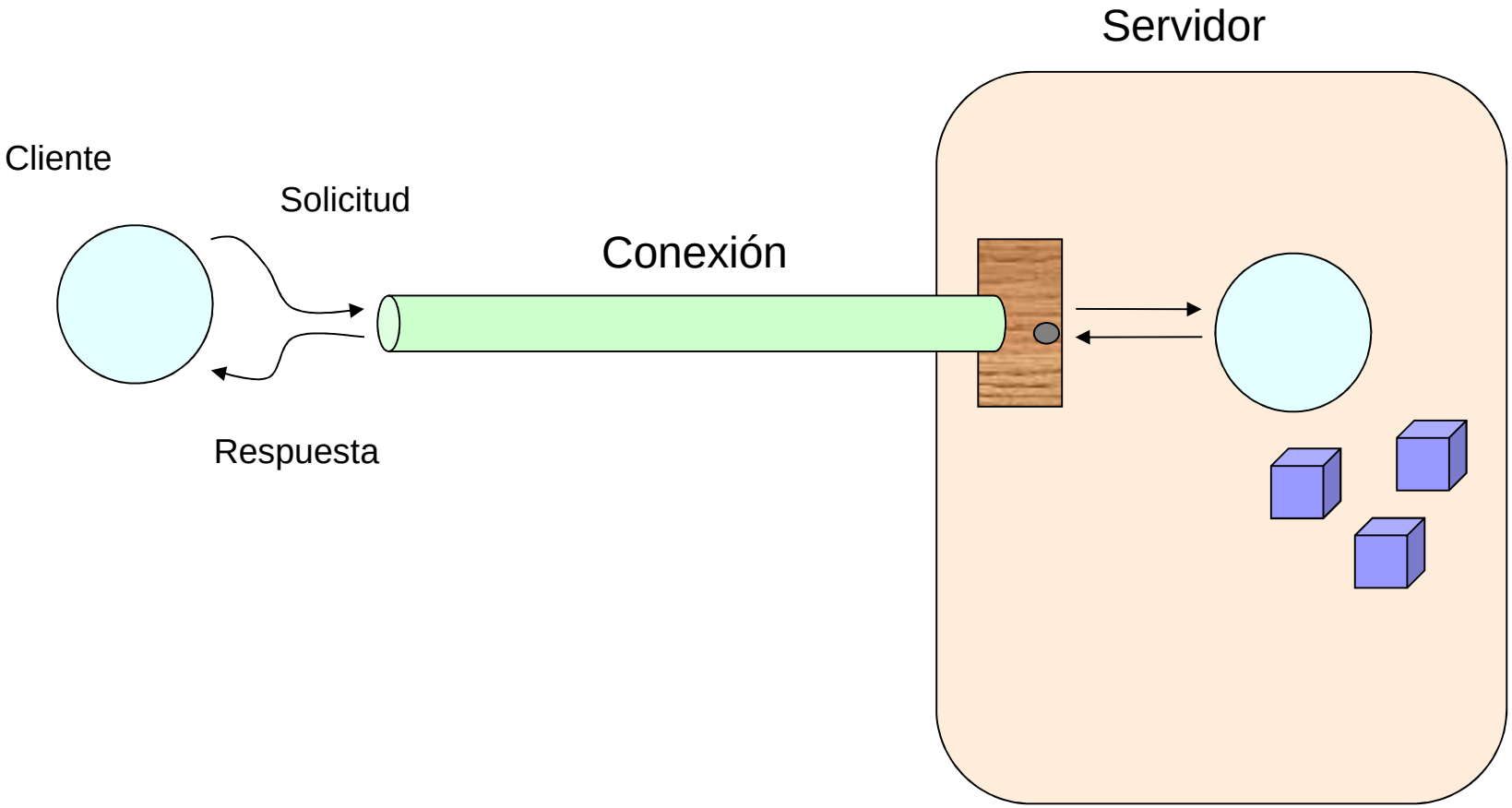
Prof. Ricardo González

Modelo de Programación Cliente-Servidor

- **Cliente:** un programa que envía peticiones**.**
- **Servidor:** un programa que ofrece un servicio que satisface peticiones que provienen de una serie de clientes**.**

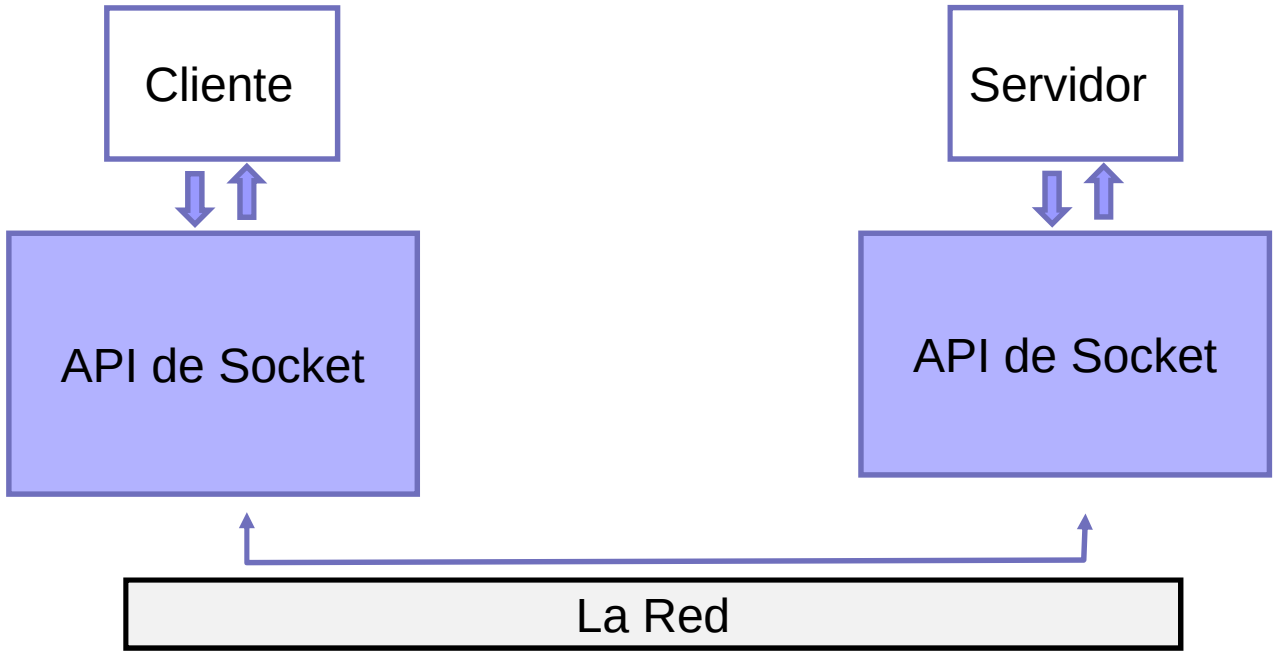
La mayoría de servicios de la Internet, tales como la Web, ftp o telnet están basados en el modelo cliente-servidor.

Modelo de Programación Cliente-Servidor que usa un protocolo orientado a conexión



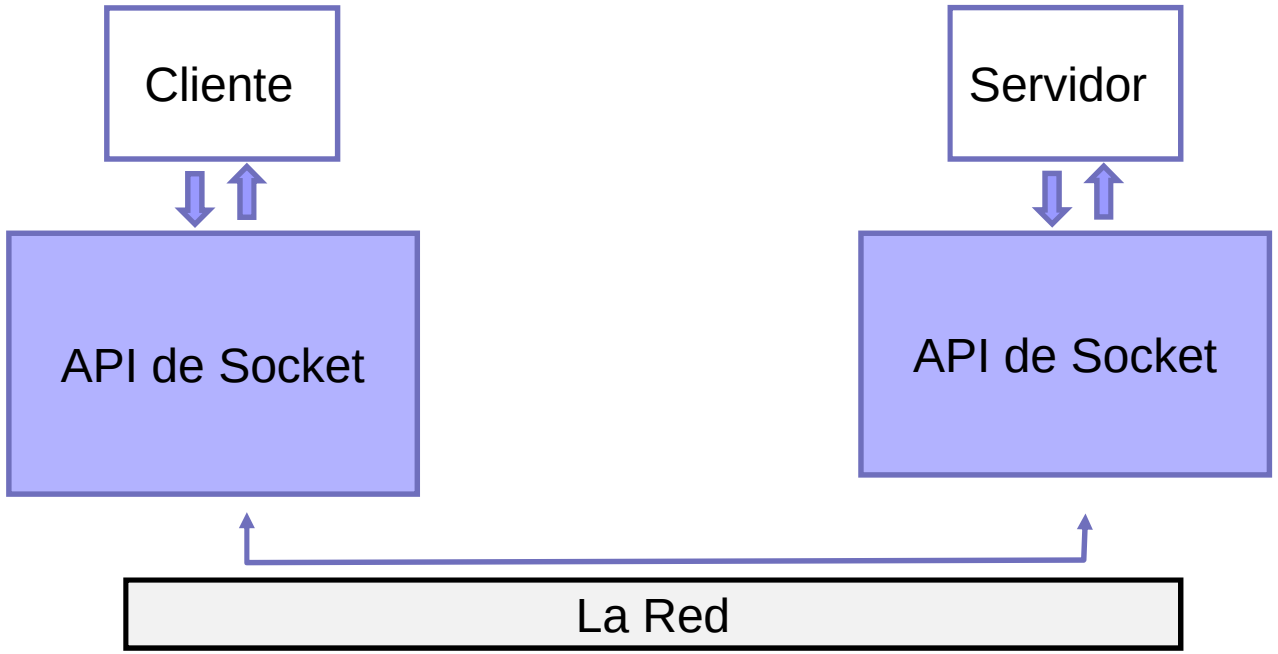
Modelo de Programación Cliente-Servidor y los Sockets

- La mayoría de las comunicación entre procesos siguen el esquema Cliente/Servidor, donde el Cliente y el Servidor son dos procesos independientes que cooperan.
- El Servidor y el Cliente intercambian mensajes a través de la red usando un API común de Sockets.



Modelo de Programación Cliente-Servidor y los Sockets

- Al momento de programar, es a traves de los sockets que se lee y escribe la data.



Modelo de Programación Cliente-Servidor y los Sockets

Ejemplos de programas

- Web browsers (Navegadores), ftp, telnet, ssh

¿Cómo un cliente encuentra a su Servidor?

-> Mediante una tupla de dos componentes

  - La dirección IP identifica el host o computador
  - El puerto (well-known) identifica el servicio, y de forma implícita al proceso que ofrece dicho servicio.

Ejemplos de puerto bien conocidos (well know ports)

- Puerto 7: Echo server
- Puerto 23: Telnet server
- Puerto 25: Mail server
- Puerto 80: Web server

Modelo de Programación Cliente-Servidor y los Sockets

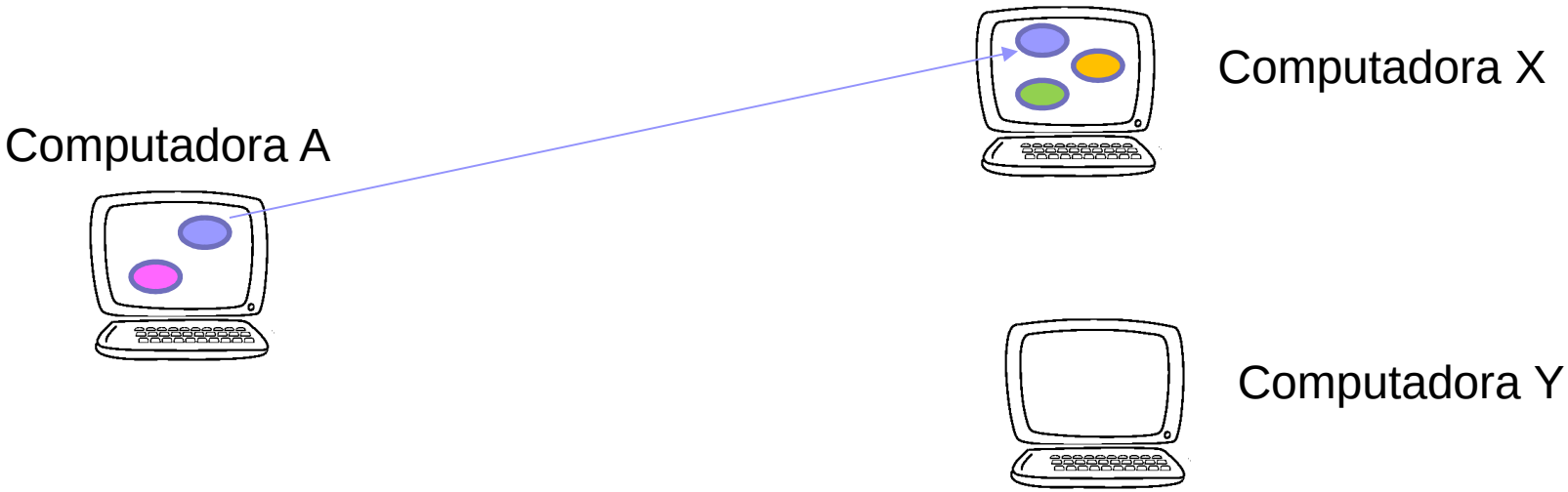
Ejemplos de programas

- Web browsers (Navegadores), ftp, telnet, ssh

¿Cómo un cliente encuentra a su Servidor?

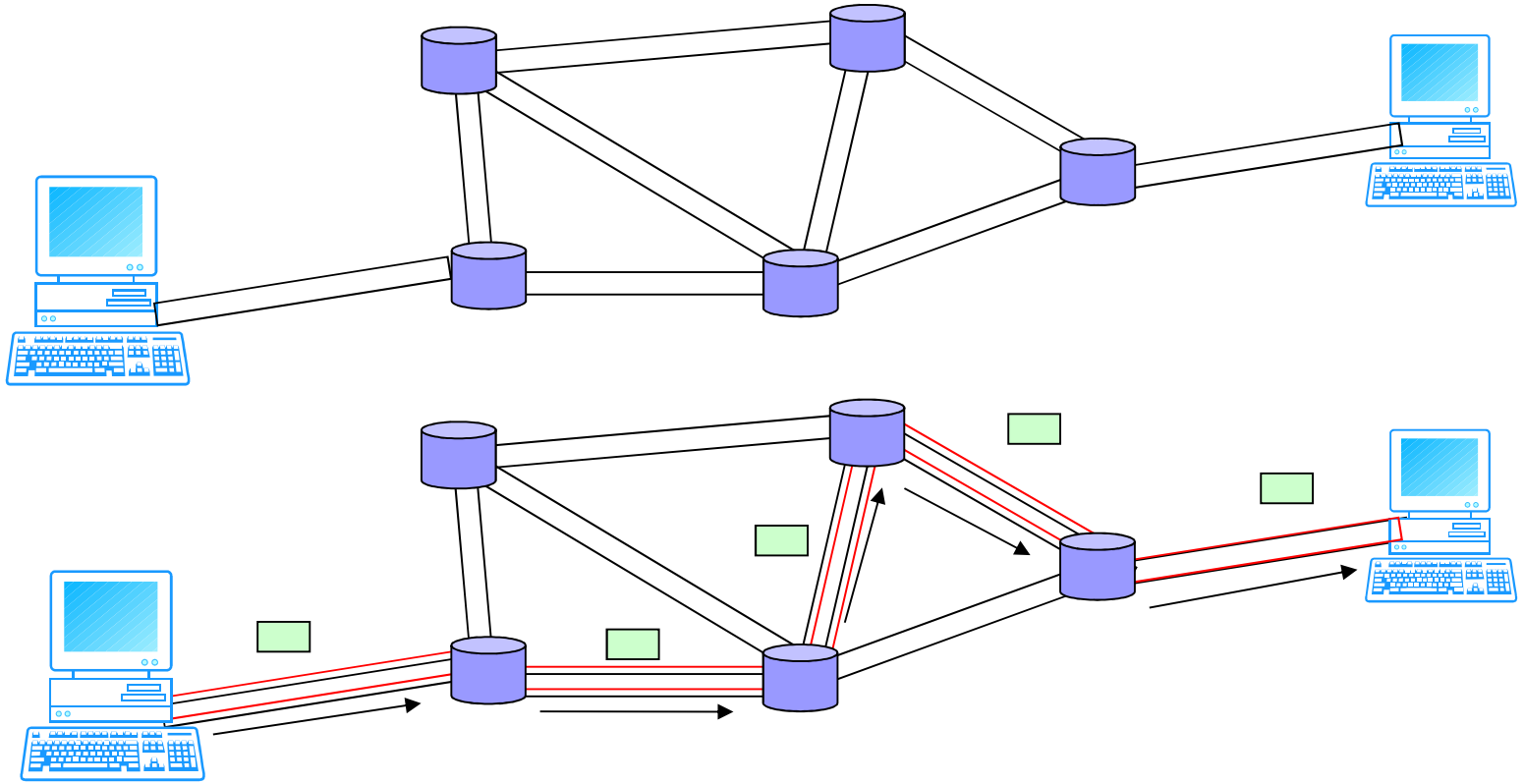
- Mediante una tupla de dos componentes

  ¿Como se comunica skype? Necesita la direccion IP y el puerto.



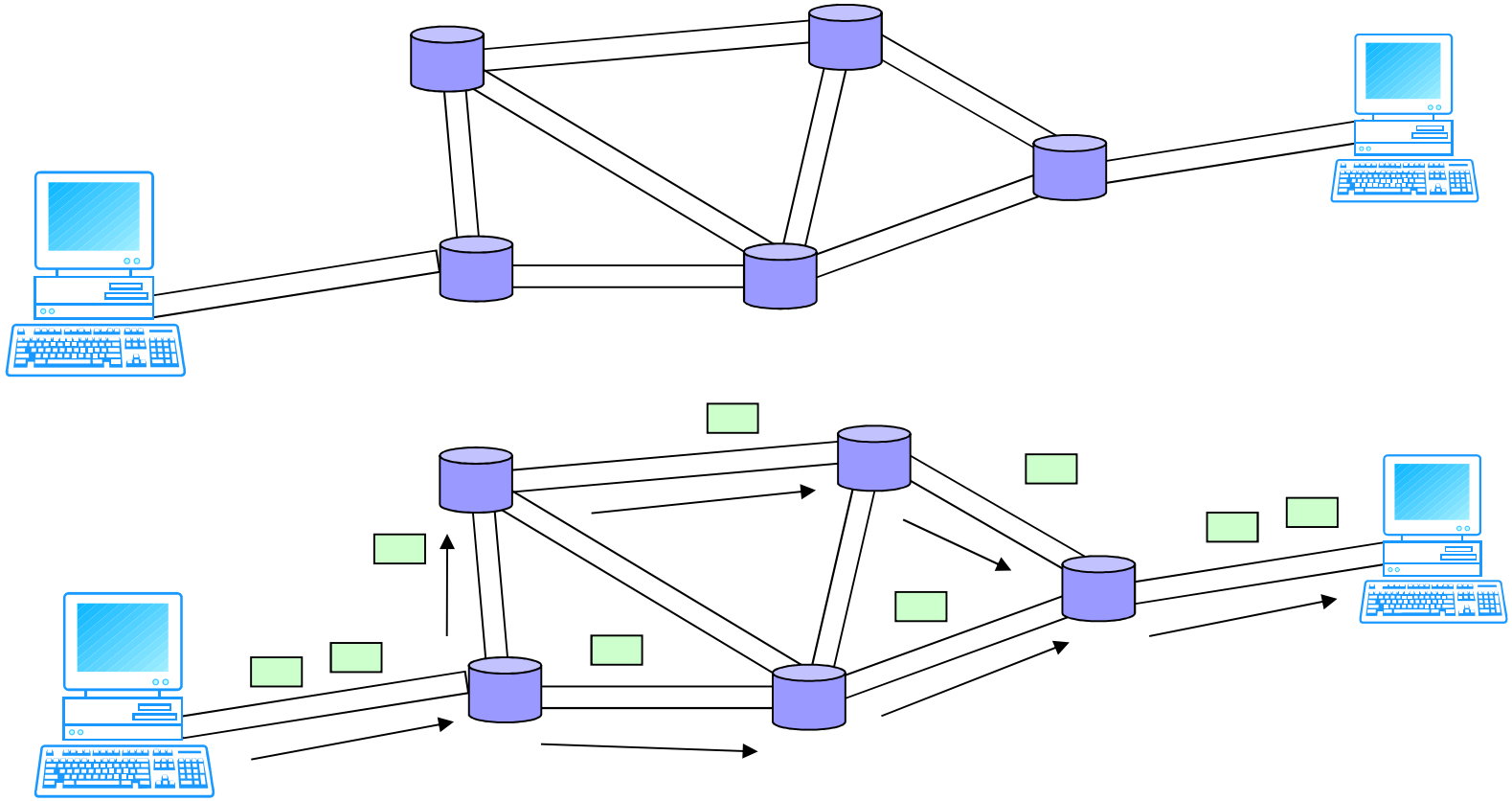
Protocolos de Comunicación

- Protocolos Orientados a Conexión



Protocolos de Comunicación

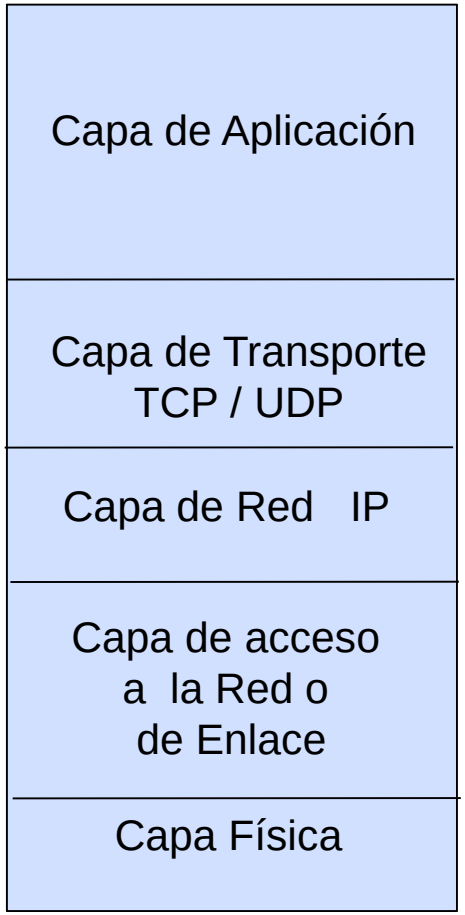
- Protocolos No Orientados a Conexión



Protocolos de Comunicación

- TCP
  - Provee una conexión confiable, para transferir bytes entre dos procesos que pueden estar en equipos distintos.
- UDP
  - Provee una transferencia no fiable de un grupo de bytes entre dos procesos.

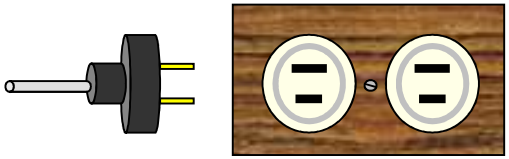
Modelo Híbrido TCP/IP

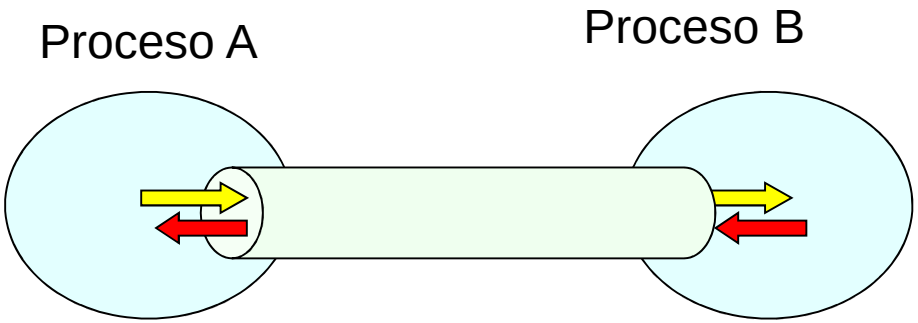


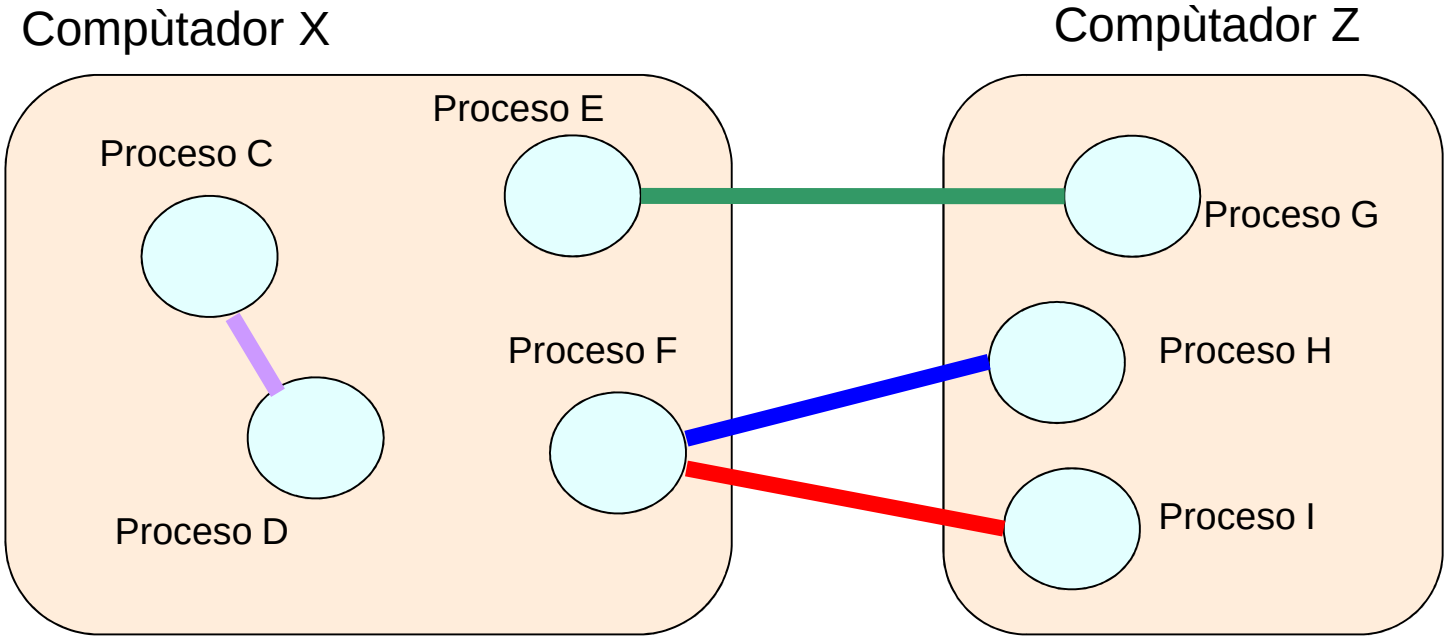
Sockets

- Los *sockets* son un API para la comunicación entre procesos que permiten que un proceso hable (emita o reciba información) con otro proceso incluso estando en distintas máquinas.
- Un **socket** es al sistema de comunicación entre computadores lo que un *buzón o un teléfono* es al sistema de comunicación entre personas: un punto de comunicación entre dos agentes (procesos o personas respectivamente) por el cual se puede emitir o recibir información. La forma de referenciar un socket por los procesos implicados es mediante un **descriptor** del mismo tipo que el utilizado para referenciar archivos.
- Se podrá realizar redirecciones de los archivos de E/S estándar (descriptores 0,1 y 2) a algun sockets y así combinar aplicaciones de la red.
- Todo nuevo proceso creado heredará, los descriptores de archivos y los sockets de su padre.

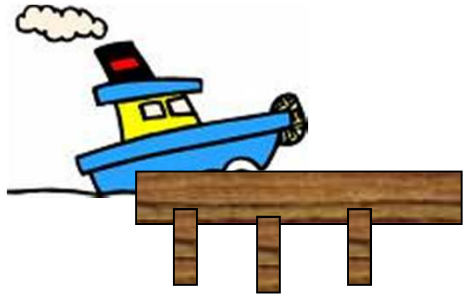
Sockets




Proceso A
Proceso B


Compùtador X
Compùtador Z
Proceso E
Proceso C
Proceso G
Proceso F
Proceso H
Proceso I
Proceso D

Puerto



(dirección, puerto)

Compùtador X

Compùtador Z

Proceso E

Proceso C

Proceso G

Proceso F

Proceso H

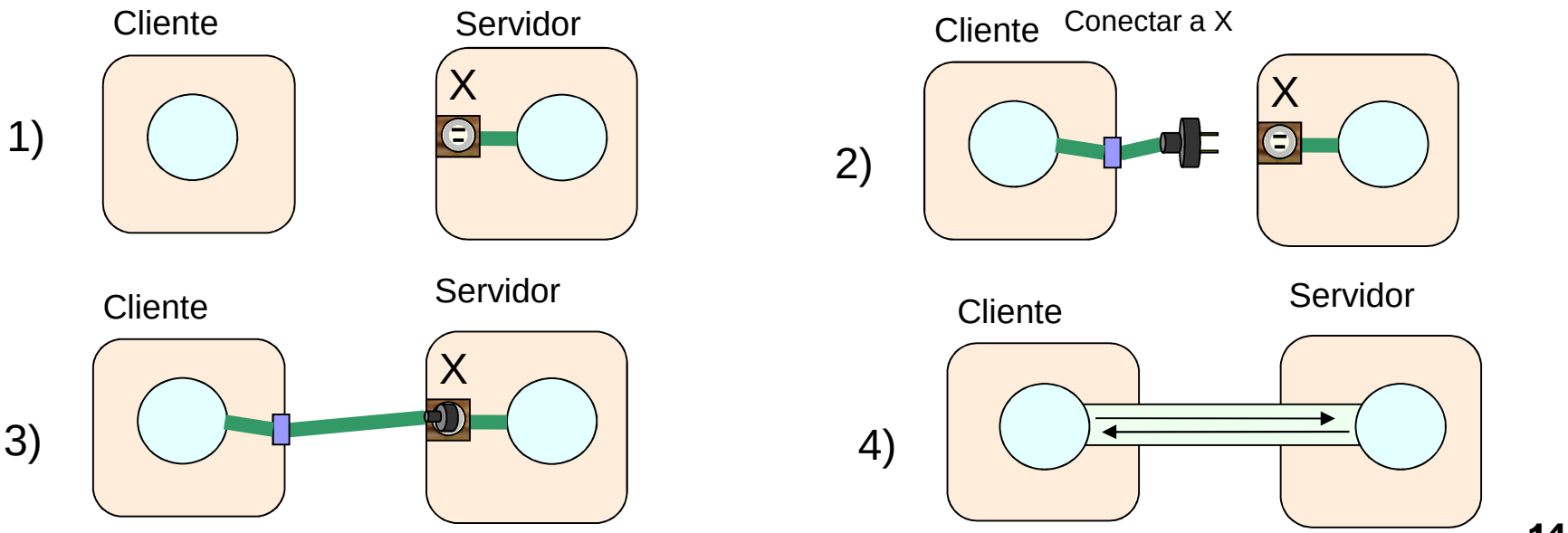
Proceso D

Proceso I

Sockets

Los pasos, en general para comunicarse vía sockets son:

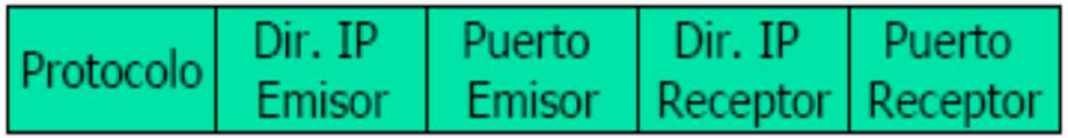
1) El servidor crea un socket cuyo nombre conocen otros procesos. Abre una conexión a un puerto bien conocido. Un puerto es un concepto lógico para saber a qué proceso dirigir la petición del cliente.
2) El cliente crea un socket sin nombre y pide una conexión al socket del servidor (con el puerto bien conocido).
3) El servidor acepta la conexión.
4) El cliente y el servidor intercambian información.



Comunicación mediante Socket

Para establecer una comunicación en red debemos conocer:

- **Protocolo de comunicación** (TCP o UDP).
- **Dirección del emisor** (dirección IP + puerto).
- **Dirección del receptor** (dirección IP + puerto).



Tipos de Servidores: Seriales

- **Seriales:** reciben la petición y hasta que no terminan con la misma no atienden una nueva petición.

```
for(;;) {
        listen for client request
        create a private two-way channel
        while (no error on communication channel)
            read request
            handle request
        close communication channel
}
```

Si el servidor es muy solicitado y las peticiones son largas (transferencia de archivos) los clientes tendrán que esperar para ser atendidos lo cual puede llegar a ocasionar inconvenientes.

Tipos de Servidores: Concurrentes

Padre-Hijo: (Servidor concurrente) el hijo maneja la petición del cliente mientras el padre sigue oyendo requerimientos adicionales.

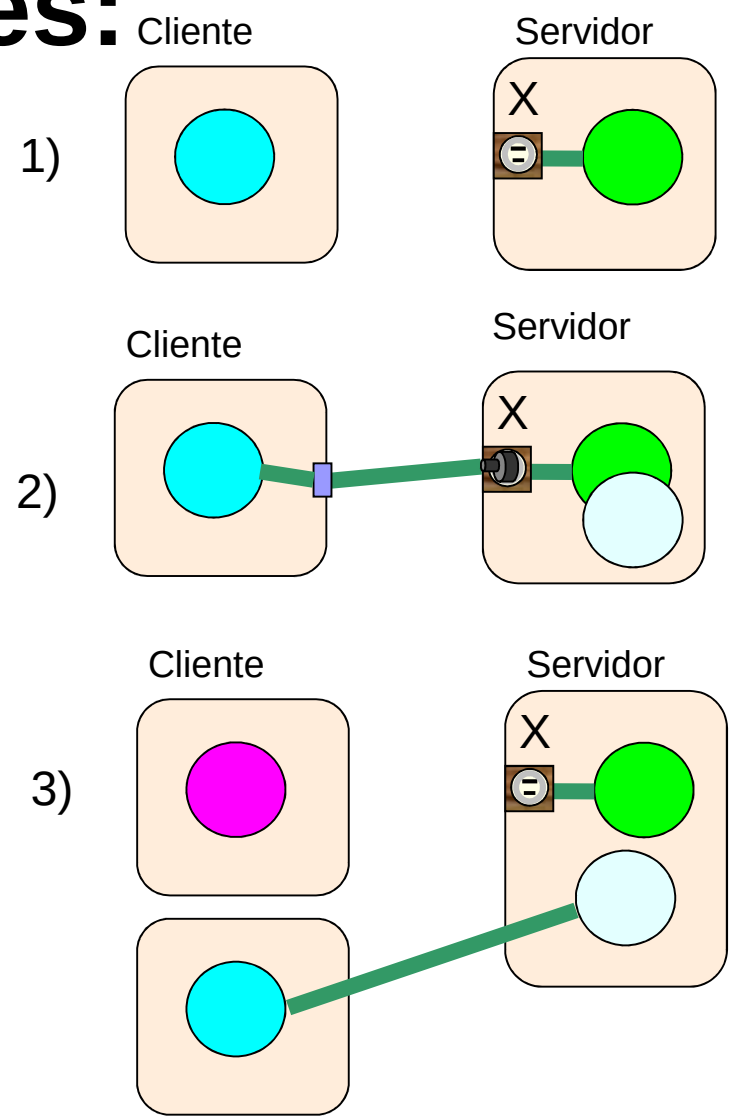
```
for(;;) {
    listen for client request
    create a private two-way channel
    if (!fork)
        handle the request
        exit
    else
        close the private channel
    clean up zombies
}
```

Servidor Multithreaded: es la estrategia de menor overhead. El servidor crea un thread en su propio espacio de direcciones. Es eficiente cuando las peticiones son pequeñas. El problema es que comparten un único espacio de direcciones y pudiera haber interferencia. Si los threads son CPU bound y no son soportados por el kernel(hilos a nivel de usuario) el servidor main thread pudiera no ejecutarse de forma concurrente.

Tipos de Servidores: Concurrentes

Padre-Hijo: (Servidor concurrente) el hijo maneja la petición del cliente mientras el padre sigue oyendo requerimientos adicionales.

```
for(;;) {
    listen for client request          1
    create a private two-way channel   2
    if (!fork)
        handle the request             3
        exit
    else
        close the private channel
    clean up zombies
}
```



Esquema general de la comunicación Cliente / Servidor

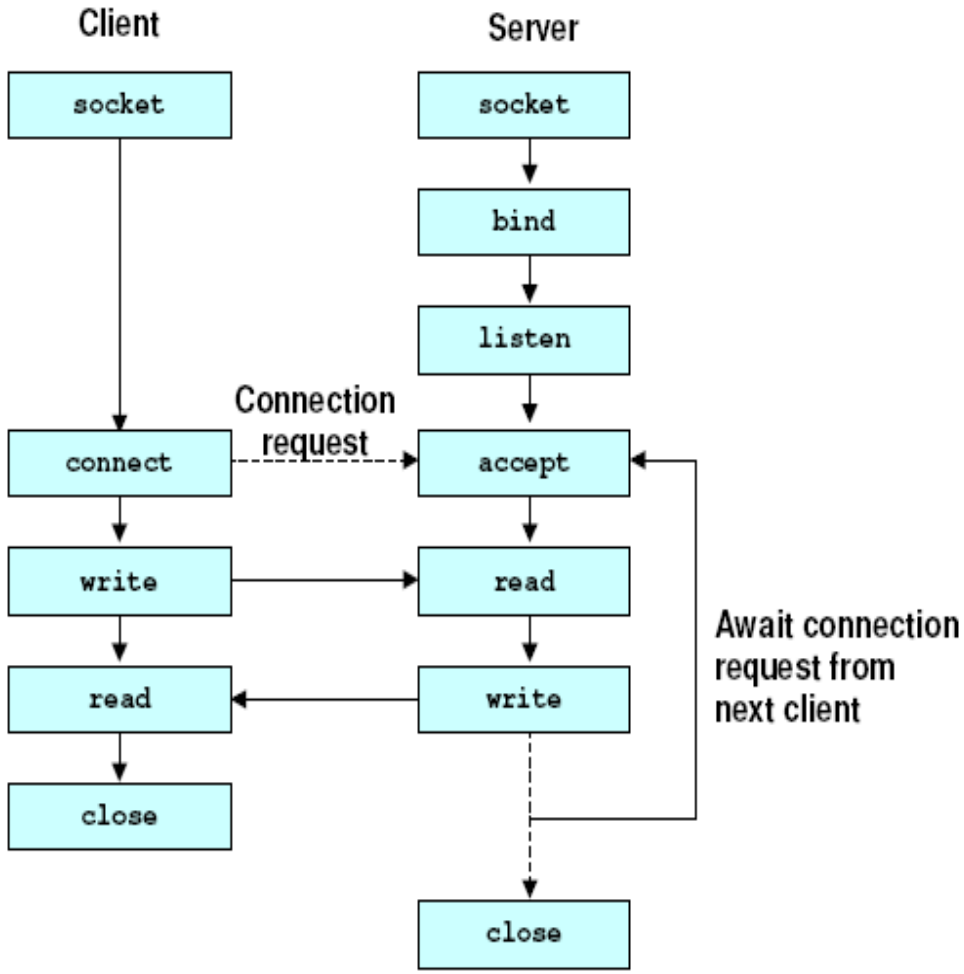
Client

1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

Server

1. Create a TCP socket
2. Assign a port to socket
3. Set socket to listen
4. Repeatedly:
   1. Accept new connection
   2. Communicate
   3. Close the connection

Esquema general de la comunicación Cliente / Servidor



Llamadas al sistema asociadas a los Sockets

Include

- Cualquier programa que emplee socket debe contar con los siguientes include:

/usr/include/sys/types.h

/usr/include/sys/socket.h

- Adicionalmente según el dominio empleado se debe contar con los siguientes include

Dominio	Archivos adicionales a incluir (include)
AF_UNIX	/usr/include/sys/un.h
AF_INET	/usr/include/netinet/in.h /usr/include/arpa/inet.h /usr/include/netdb.h

Creación de un Socket

int socket (int domain, int type, int protocol):

- permite crear un socket no nominado y *retorna un descriptor de archivo.* El socket se liga a un protocolo particular pero no se conecta a nada. Hasta que no está conectado no se puede leer de él o escribir en él.
- La llamada socket, al igual que la llamada Open(), retorna un descriptor de archivo si la llamada ha tenido éxito o un valor que es menor que cero si ocurre un error (como -1). Después que se conecta el socket, el descriptor de archivo puede ser utilizado para leer (read) o escribir (write).
- Cuando un proceso termina de usar el socket debe cerrarlo para liberar los recursos asociados a él.

- Siempre verificar si la operacion fue excitosa o no!!!!

Creación de un Socket

int socket (int domain, int type, int protocol):

Parámetros:

- El **dominio**: indica dónde puede residir el socket del servidor y del cliente. Es el dominio donde las direcciones serán interpretadas. El dominio tiene una familia de protocolos asociada.

Dominio	Descripción
PF_UNIX	Dominio UNIX: el cliente y el servidor deben estar en la misma máquina.
PF_INET	TCP/IP v4: el cliente y el servidor pueden estar en cualquier lugar en la Internet.
PF_APPLETALKAppleTalk DDS	AppleTalk DDS

Creación de un Socket

int socket (int domain, int type, int protocol):

Parámetros:

El tipo: Es el tipo de comunicación que puede existir entre el cliente y el servidor. Hay varios tipos pero sólo dos son de interés:

- SOCK_STREAM: circuito virtual. Garantiza un manejo confiable de los datos en el orden que son enviados. Se envían cadenas de bytes de longitud variable. No se puede enviar datos mientras el circuito no se haya establecido. El circuito permanece intacto hasta que se destruya explícitamente.
- SOCK_DGRAM: este tipo de conexión se usa para enviar distintos paquetes de información llamados datagramas. No se garantiza que lleguen en orden, no se garantiza siquiera que lleguen. Los mensajes son de longitud fija. No hay conexión privada.

Creación de un Socket

int socket (int domain, int type, int protocol):

Parámetros:

- Un **protocolo** es un juego de reglas que define la forma en que deben efectuarse las comunicaciones de las redes, incluyendo el formato, la secuencia, tamaño de mensajes, etc.
- El protocolo: está sujeto a las restricciones impuestas por los primeros dos parámetros. Su valor indica la forma de bajo nivel en la que es implementado el socket. Las llamadas al sistema que esperan como parámetro al protocolo acepta el valor 0 para indicar el protocolo correcto, dejando al kernel elegir el protocolo por defecto. Si la familia es PF-INET o AF_INET, TCP es el protocolo orientado a conexión y UDP es el protocolo no orientado a conexión.

- Ejemplo:

```
Int sockfd;
sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

Rutinas asociadas al servidor: Bind

Asociar un socket a una dirección

*int bind (int sockfd, struct sockaddr *my_addr, int addrlen)*:

- Es como asignar un nombre a un socket. Esta llamada asocia una dirección y un puerto bien conocido al socket identificado por el descriptor de archivo sockfd . Retorna 0 si tiene éxito y -1 si se presenta algún error.
- sockfd = file descriptor que retorna la llamada a socket
- *struct sockaddr **

  Las direcciones del dominio UNIX son pathnames del file system. Estas direcciones se pasan a través de la siguiente estructura:

```
struct sockaddr_un {
        unsigned short sun_family;              /* AF_UNIX */
        char sun_path[UNIX_PATH_MAX]        /* pathname */
}
```

  Ejecuta: $man bind para ver el ejemplo

- *addrlen:* Longitud de la estructura contenida en el parámetro 2 (el anterior)

Esta se puede calcular usando sizeof(*my_addr)

Rutinas asociadas al servidor: Bind

- Las direcciones IP empleadas por el dominio INET se cargan en una estructura con el siguiente formato:

```
struct sockaddr_in {
    short int sin_family;              /* AF_INET */  AF_INET
    unsigned short int sin_port        /* port number */ htons(port_number)
    struct in_addr sin_addr            /* IP address */ Nota que es una estructura.
}
```

- El primer elemento indica que es una dirección IP y no el nombre de un archivo.
- **El segundo elemento es el número de puerto**. Los puertos son números de 16 bits que identifican unívocamente el punto final de una conexión a un hosts. La combinación de una dirección IP y un número de puerto identifican el punto final de una conexión en cualquier lugar de una red TCP/IP. Aunque los números de puerto van del 0 al 65,535, Linux la divide en dos clases. Los puertos reservados van del 0 al 1024 deben usarlo sólo los procesos o servicios estándares del sistema. El puerto debe ir en el orden correcto de bytes.
- El número final es la **dirección IP** de la máquina donde el puerto reside.

Direcciones IP v4

- Las direcciones IP: cada nodo en una red TCP-IP requiere de una dirección numérica de 4 bytes o 32 bits. Lo más común es escribirla en notación decimal, lo cual se obtiene transformando cada byte a su forma decimal:

11000000 10101000 00000001 0001101

192 168 1 25

192.168.1.25

Direcciones IP v4

- Las direcciones IP están asociadas a nombres. Estos nombres se encuentran en una base de datos. A estas BD se accede a través de los servidores de nombres distribuidos por Internet. La base de datos se llama DNS o Domain Name System.
- El *mapping* entre nombres y direcciones IP es *many to many*. Varios *sites* o servicios pueden usar una misma máquina, por ejemplo el ftp y el web. Pueden tener dos nombres www.some.org y ftp.some.org y sólo una dirección IP si se ejecutan en una misma máquina. Por su parte un router puede tener varias direcciones IP y un solo nombre.

Direcciones especiales.

- 127.0.0.1 local host (el mismo computador donde se esta trabajando)
- Direcciones que terminan en 0 = una subred
- Direcciones que terminan en 1= un router.

Estableciendo el número máximo de clientes en espera: Listen

int listen(int sockfd, int backlog)

- Mientras un proceso servidor está dando servicio a un cliente conectado, siempre es posible que otro cliente intente establecer una nueva conexión con el servidor. La llamada al sistema listen permite especificar al proceso servidor el número de conexiones pendientes que deben ser encoladas antes de que la siguiente solicitud sea descartada. Luego de sobrepasar este valor el siguiente cliente que intente conectarse fallaría en su llamada *connect*.
- sockfd: file descriptor
- backlog: cuantas conexiones pueden estar pendientes esperando ser aceptadas. Históricamente se ha colocado el valor de 5, aunque puede colocarse otro valor.
- Listen retorna 0 si tiene éxito y un valor diferente de 0 si falla.

- Ejemplo: listen(sockfd, num); #num es una constante

Aceptación de la conexión de un cliente

*int accept(int sockfd, struct sockaddr *addr, int addrlen)*

- Una vez que un socket es creado, nominado y el tamaño de la cola de clientes establecido, el paso final es aceptar el requerimiento de conexión de un cliente. Esto lo puede hacer el servidor mediante la llamada accept.
- El servidor acepta la conexión por un socket. Al llegar a este punto normalmente el servidor detiene su ejecución en espera de que un cliente establezca una conexión.
- La llamada *accept* es bloqueante, a menos que se haya especificado no bloqueante en la creación del socket.
- La llamada al sistema retorna un nuevo socket no nominado, asociado al descriptor de archivo sockfd, que representa al socket que debe usarse para la conexión con el cliente. Este nuevo descriptor hereda los atributos del socket por donde se hizo el listen.
- Los parámetros *addr y addrlen* *apuntan a datos del cliente que el kernel llena.* Al crearse el nuevo socket para el cliente, el socket original puede volver a ser empleado para esperar nuevas conexiones de otros clientes.
- Accept al igual que open retorna un descriptor de archivo si tiene éxito o un número menor que 0 si ocurre un error. Inicialmente *Addrlen* debe ser un entero que contiene el tamaño de la estructura *sockaddr*.

Rutinas asociadas al Cliente

- Para poder establecer la comunicación vía socket con un servidor, el proceso cliente debe llenar una estructura con la dirección del servidor y el puerto, para luego emplear la llamada connect para intentar establecer la conexión con el servidor. Estos datos se suelen dar via un archivo.

  *int connect (int sockfd, struct sockaddr *seraddr, int addrlen)*

- Si la conexión es exitosa la rutina devuelve el valor 0 y el valor sockfd puede ser empleado como descriptor de archivo para comunicarse con el servidor, mediante las llamadas read y write. Si el socket del servidor no existe o su cola de procesos en espera esta llena, la llamada connect retorna el valor -1, para indicar que la conexión no fue exitosa.

Rutinas asociadas al Cliente

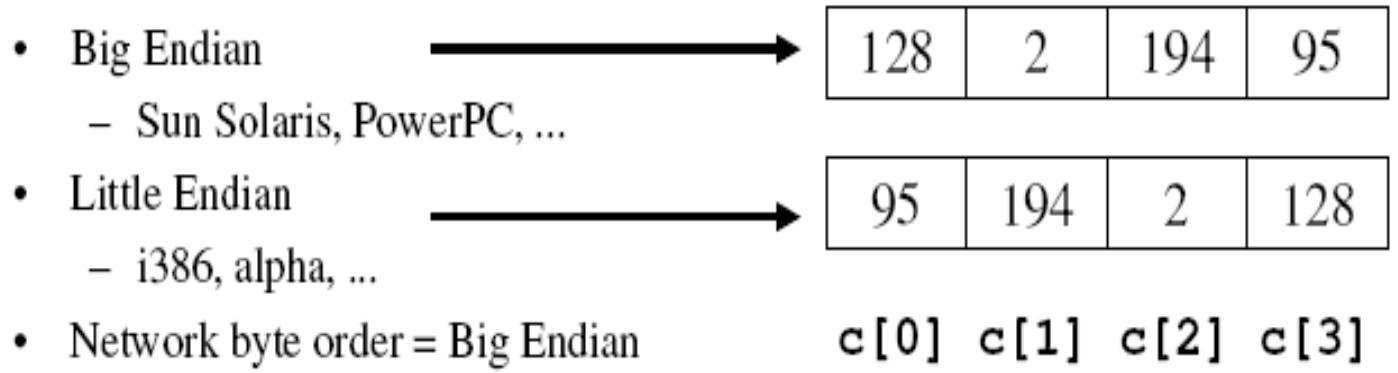
*int connect (int sockfd, struct sockaddr *seraddr, int addrlen)*

- Los parámetros son el socket del cliente y la dirección a la cuál el cliente se desea conectar pasada a través de la estructura *seraddr*.
- Si el socket pertenece al dominio AF_UNIX, el parámetro es un apuntador a una estructura *sockaddr_un, en este caso* se le debe hacer un *cast* a *sockaddr**, para ser pasado como *seraddr*.
- Si el socket pertenece al dominio AF_INET, el parámetro es un apuntador a una estructura *sockaddr_in, en este caso* se le debe hacer un *cast* a *sockaddr**, para ser pasado como *seraddr*.
- El parámetro *addrLen* debe ser igual al tamaño de la estructura referenciada por *sockaddr*.

Otras Rutinas asociadas a la comunicación de datos

Orden de los bytes

- Las redes TCP-IP son heterogéneas e incluyen una gran variedad de máquinas y arquitecturas. Una de las diferencias más comunes en las arquitecturas es cómo almacenan los números. Big endian carga el byte más significativo en la dirección más baja y little endian hace justo lo contrario.
- TPC/IP adopta big-endian para transmitir datos o información del mismo protocolo y sugiere también que se le aplique a los datos. Este orden se conoce como network byte order.
- Existen una serie de funciones para convertir del orden en el cual son trabajados los números en el computador que ejecuta el código (*host*) al orden de la red (*network*) y viceversa.



Otras Rutinas asociadas a la comunicación de datos

Todos los valores almacenado en **sockaddr_in** deben estar en network byte order.

Se usa esta estructura para la comunicacion entre sockets!

- **sin_port** El número del puerto TCP/IP.
- **sin_addr** La dirección IP. Cuidado que este es un struct! La IP esta en uno de los campos. **sin_addr.s_addr**

unsigned int htonl (unsigned int hostlong)

- *Host to network long* convierte un entero largo (32 bits), que esta originalmente en el formato de la máquina local, al formato de red.

unsigned short htons (unsigned short hostshort)

- *Host to network* short convierte un entero corto (16 bits), que esta originalmente en el formato de la maquina local, al formato de red.

unsigned int ntohl (unsigned int netlong)

- *Network to host long* convierte un entero largo (32 bits), que esta originalmente en el formato de red, al formato de la maquina local

unsigned int ntohs (unsigned short netshort)

- *Network to host short* convierte un entero corto (16 bits), que esta originalmente en el formato de red, al formato de la maquina local
- Estas rutinas trabajan con cantidades con signo y sin signo. Aunque se usa el término long las rutinas esperan cantidades de 32 bits.

Manejo de las direcciones IP

*char * inet_ntoa(struct in_addr address)*

- Toma una estructura del tipo in_addr que contiene la dirección IP binaria (32 bits) y retorna un string conteniendo la dirección en formato decimal (A.B.C.D).

int *inet_aton(const char *daddress, struct in_addr *address)*

- Espera un string que contiene la dirección IP en decimal y llena la estructura *address* con la representación binaria. Retorna 0 si ocurre un error y non-zero si la conversión fue exitosa.

unsigned long *inet_addr(char* dirip)*

- A partir de la variable *dirip* que contiene la dirección IP de un computador en el formato A.B.C.D retorna la direccion IP en un formato de 32 bits, que está en network-byte order.

Manejo de las direcciones IP

Nombres de hosts (Para el cliente)

Para manejar la complejidad del mapping entre direcciones IP y nombres se definen la siguiente estructura:

```
struct hostent {
    char *h_name;
    char **h_aliases;
    int h_addrtype;
    int h_length;
    char **h_addr_list;
}
```

char *h_name: nombre oficial del host.

char **h_aliases: alias para el host

int h_addrtype: tipo de dirección (AF_INET en Ipv6 se verán otros tipos de direcciones)

int h_length: longitud de la dirección binaria sizeof (struct in_addr)

char **h_addr_list: arreglo con direcciones de este host. Cada posición del arreglo apunta a algo del tipo struct in_addr.

char *h_addr

This is a synonym for h_addr_list[0]; in other words, it is the first host address.

Manejo de las direcciones IP

- Dos funciones de librería convierten entre números IP y hostnames.

Gethostbyname: retorna un struct hostent para un hostname que contiene la dirección IP de la máquina especificada en el parámetro name.

Gethostbyaddr: retorna información sobre una máquina con una dirección IP en particular.

*Struct hostent *gethostbyname(const char *name);*

*Struct hostent *gethostbyaddr(const char *addr, int len, int type):* el primer parámetro debería apuntar a un struct in_addr. El parámetro *len* es la longitud de la estructura y *type* especifica el tipo de dirección.

- Cuando ocurren errores buscando un nombre de host se establece el valor de la variable h_errno. Se usa entonces herror() que imprime la descripción de error.
- Retornan un apuntador a una estructura que puede sobreescribirse cuando se vuelva a llamar a la función. El programa debería salvar valores que vaya a necesitar después.

Manejo de las direcciones IP

La dirección del host donde se ejecuta el programa se puede obtener mediante la rutina gethostname.

int gethostname (char name, int nameLen);*

Esta rutina llena el arreglo de caracteres apuntado por name, de longitud nameLen con un string terminado en null que contienen la dirección del host local donde se ejecuta el programa.

Fin primera Clase

Mostrar Ejemplos del Profesor Ricardo.