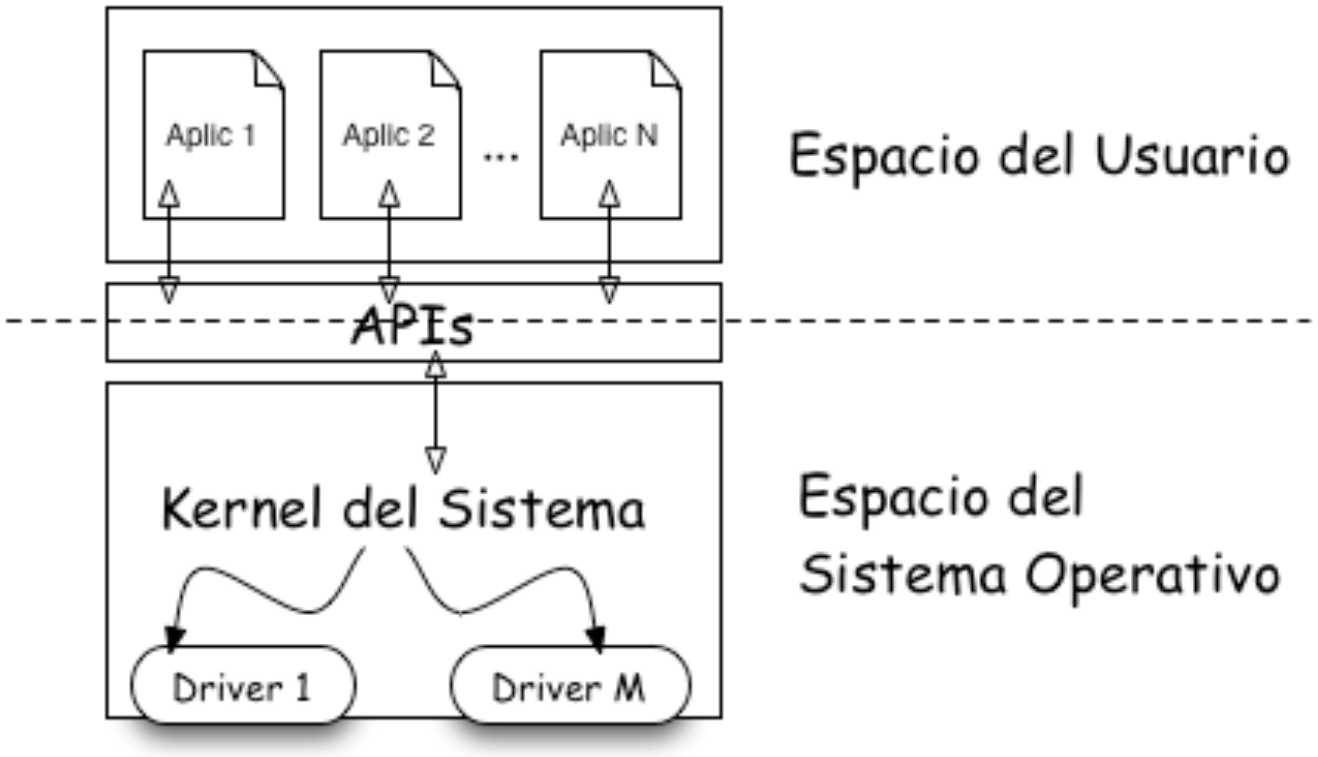
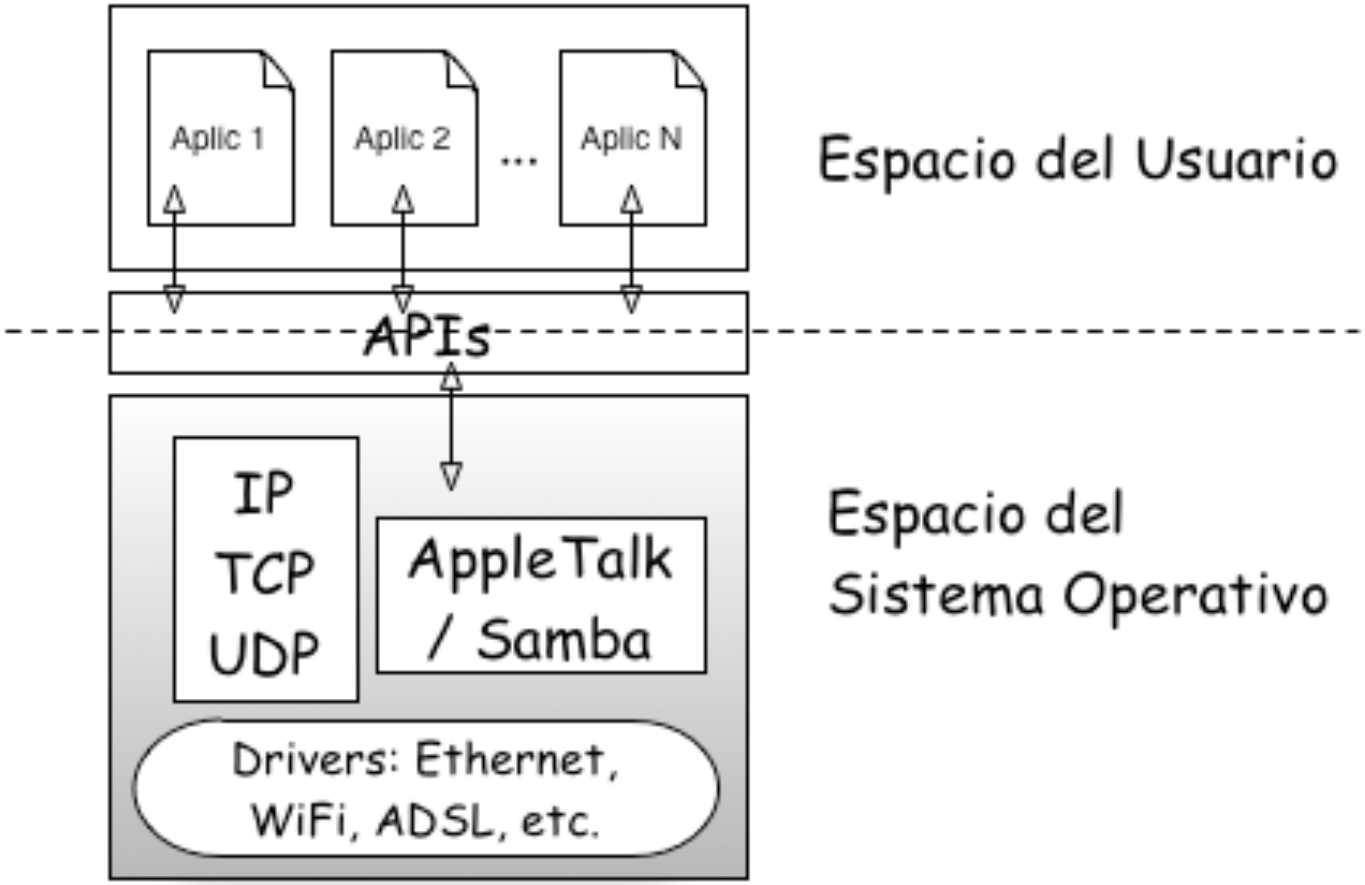
INTRODUCCION A SOCKETS

Andrés Arcia

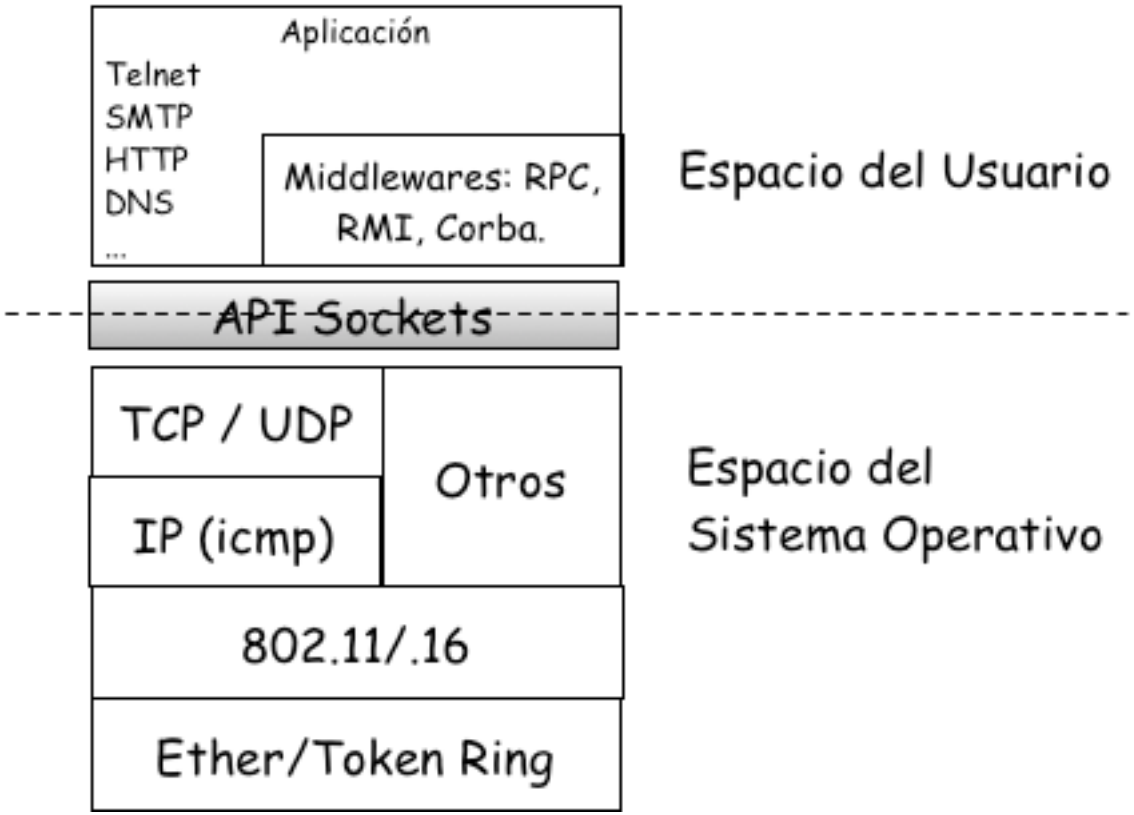
Configuración de las Aplicaciones



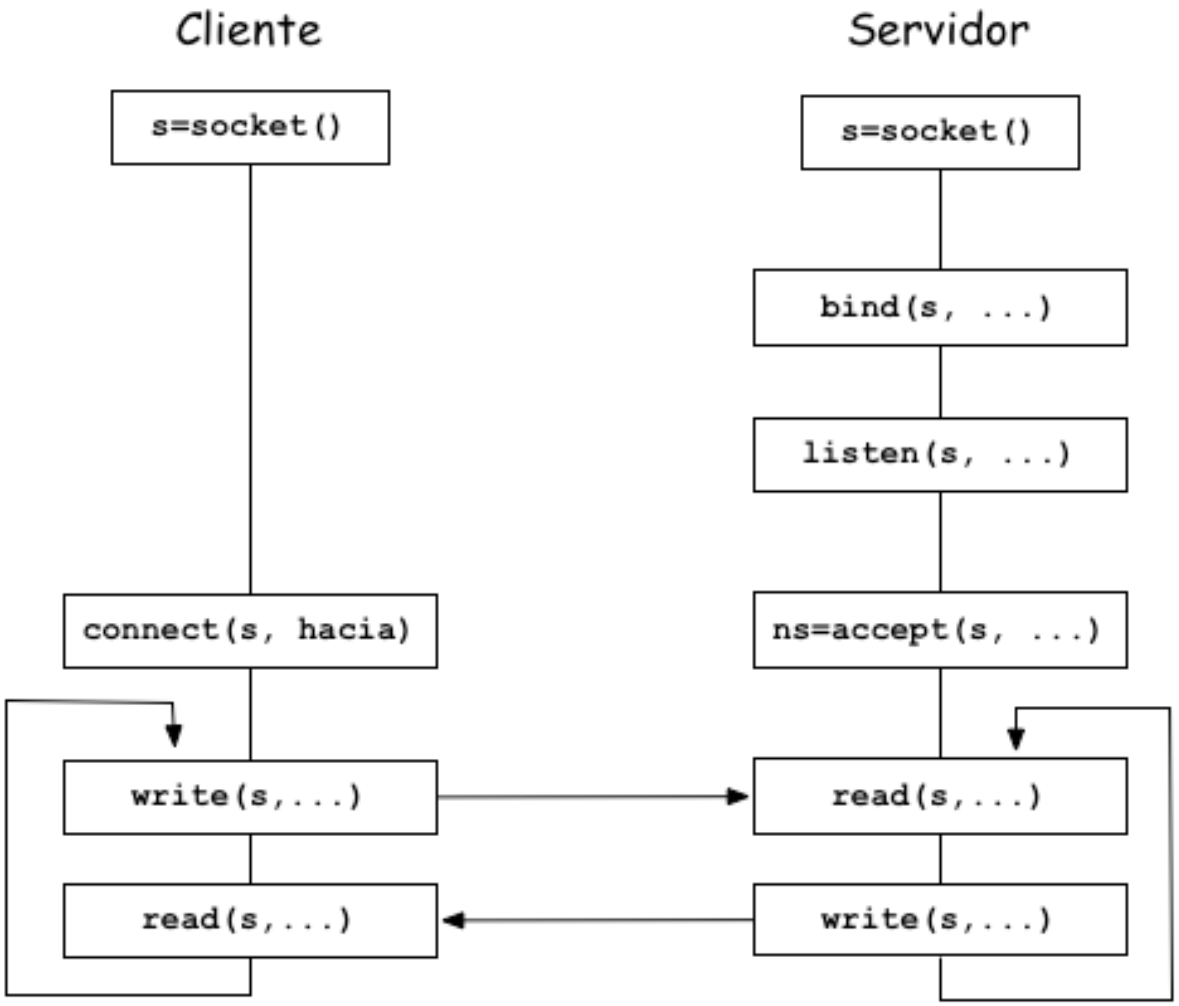
¿Dónde están los protocolos?



Arquitectura de protocolos Inet



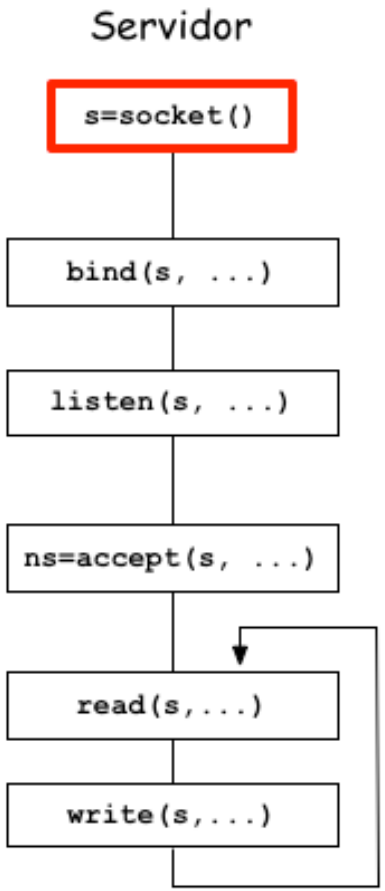
Modelo Cliente-Servidor a través de Sockets



La función sockets()

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int dominio, int tipo, int protocolo)
```

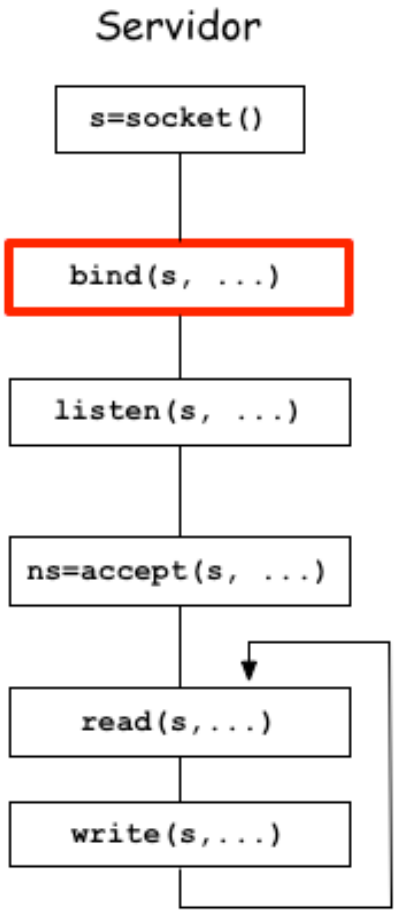
- Es el punto de acceso a la pila protocolar desde las aplicación.
- El valor de retorno (int) corresponde al identificador único del socket.
- Dominio: PF_UNIX, PF_INET, PF_INET6
- Tipo: SOCK_STREAM, SOCK_DGRAM, SOCK_RAW
- Protocolo: Permite definir el protocolo si no puede ser determinado con el Dominio y Tipo.



La función Bind

```
int bind(int sockfd,
         struct sockaddr *mi_direccion,
         socklen_t addrlen);
```

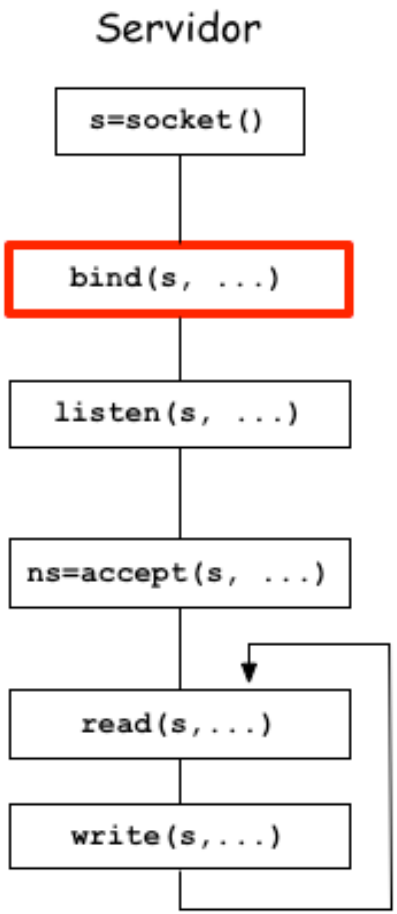
- Permite la asociación entre una dirección (ej: IP) y el identificador único otorgado por **socket** (ingresado en sockfd)
- El formato de la dirección depende del tipo de dirección seleccionada PF_UNIX, PF_INET, PF_INET6.
- El tipo específico del segundo argumento dependerá de la función donde se utilice.



Ejemplo de la Función Bind

```
struct sockaddr_in sin;
int s, puerto, r;
…
s = socket(PF_INET, SOCK_STREAM, 0);

puerto = 5690;
sin.sin_family = AF_INET;
sin.sin_addr.s_addr = INADDR_ANY;
sin.sin_port = htons(puerto);
r = bind(s, (struct sockaddr*)&sin, sizeof sin);
if (r < 0) {
   perror("error de bind");
   …
}
```

Tipo forzado (casting)



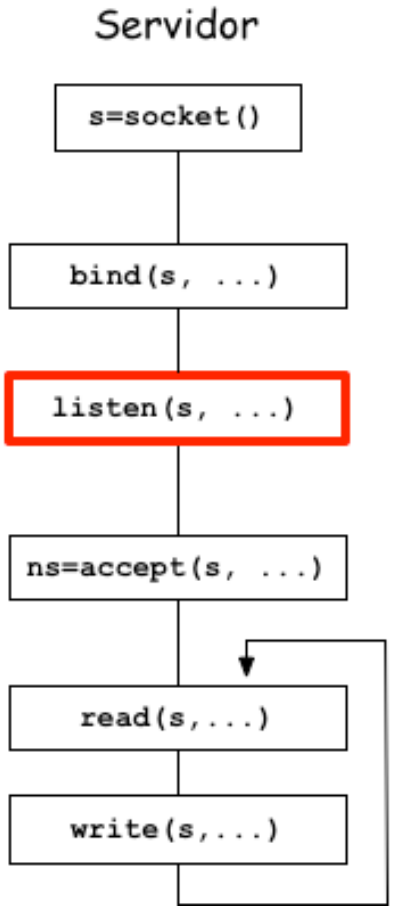
La estructura **sockaddr_in**

```
typedef uint32_t in_addr_t;

struct in_addr
{
   in_addr_t s_addr;
}

struct sockaddr_in
{
  sa_family_t sin_family;
  in_port_t sin_port;      // numero de puerto
  struct in_addr sin_addr; // direccion Inet
};
```

La función listen()

```
int listen(int s, int max_clientes)
```

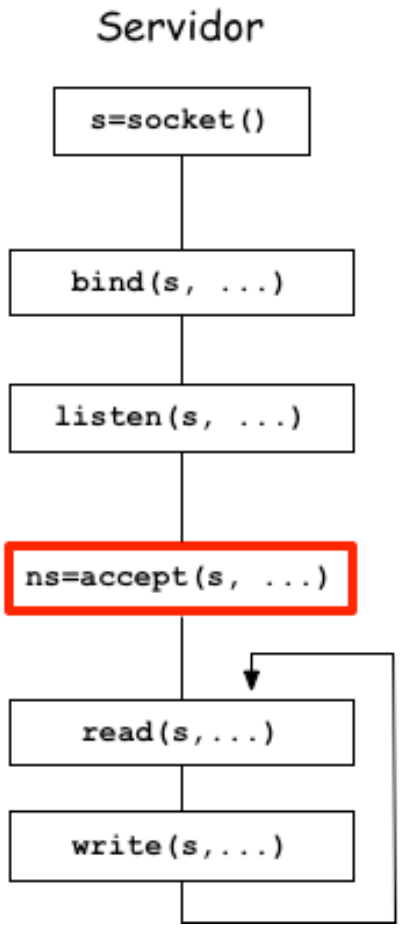
- Es la función encargada de colocar al **socket** s en modo servidor.
- La única función de esta llamada es esperar por solicitudes y actuar según la máquina de estados de TCP. *No se ocupa del intercambio de datos.*
- El parámetro "max_clientes" dice el máximo de solicitudes en cola que puede atender, antes de recharzarla. Recordemos que atender una solicitud lleva un tiempo.



La función accept()

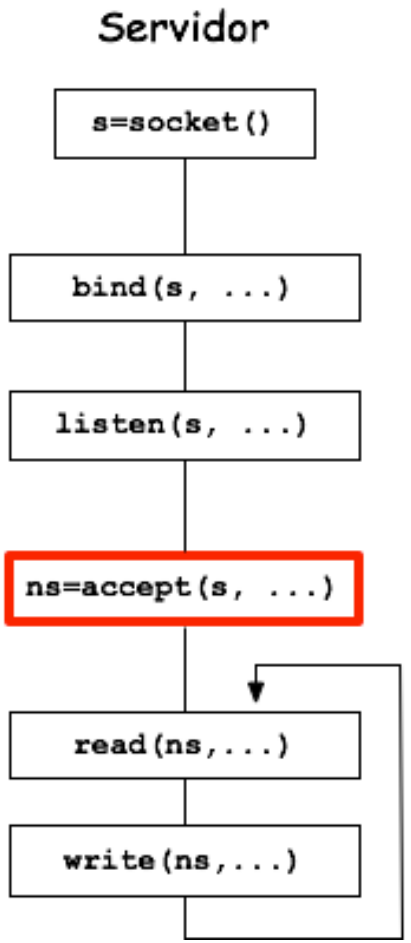
```
int accept(int s,
           struct sockaddr *addr,
           socklen_t *addrlen);
```

- Esta función "bloqueante" acepta las solicitudes de conexión, es decir, aparta los recursos necesarios para manejar la conexión.
- El parámetro **addr** guarda la dirección del socket remoto que quiere conectarse.
- El parámetro **addrlen** corresponde a la longitud del campo **addr**.



La función accept(): ejemplo

```
int s, ns, tamano_desde;
struct sockaddr_in desde;
…
tamano_desde = sizeof(desde);
ns = accept(s, (struct sockaddr *)&desde, &tamano_desde);
```

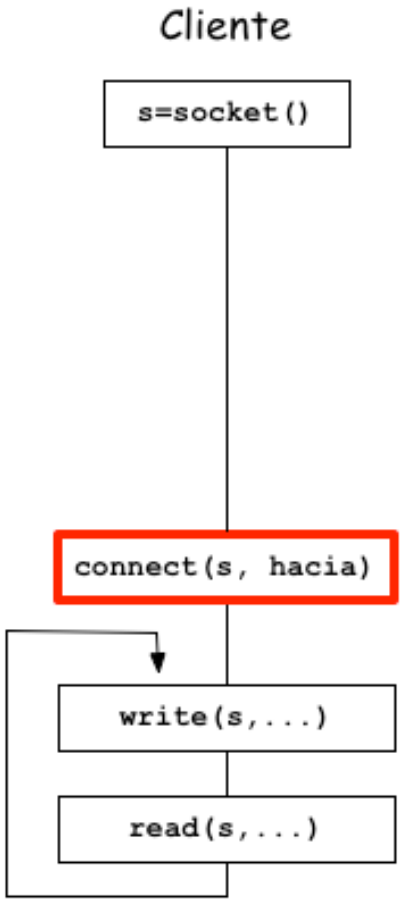


- accept() devuelve un nuevo socket que podemos memorizar en la variable **ns**. Este nuevo socket es una copia muy parecida al socket de escucha.
- **ns** sirve para el intercambio (entrada y salida) de datos.

El cliente: la función connect()

```
int connect(int s,
            const struct sockaddr *direccion_servidor,
            socklen_t addrlen);
```

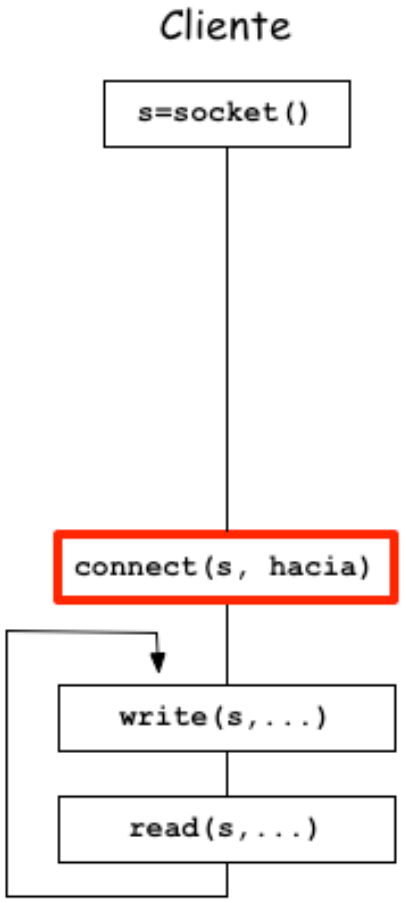
- Esta función ayuda a establecer la conexión de un socket en el cliente (s) contra un servidor especificado en el segundo parámetro (direccion_servidor).
- Recuerde que el servidor debe estar escuchando y dispuesto a otorgar una conexión para que esta función tenga éxito.



Ejemplo de connect() (cliente)

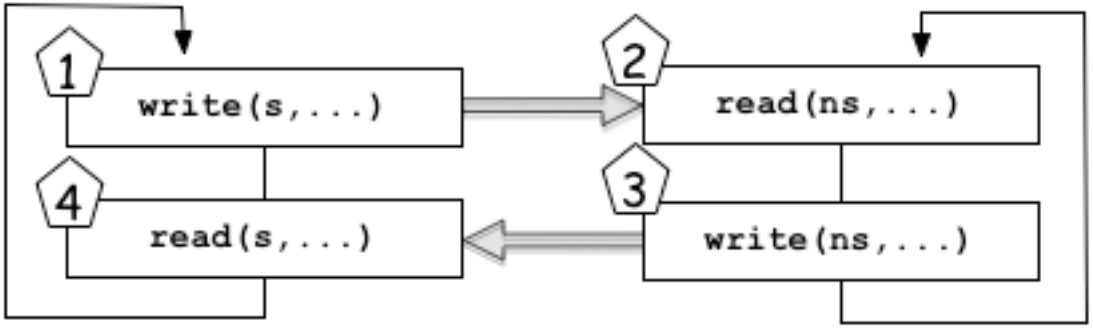
```
1. int r;
2. struct hostent *hp;
3. struct sockaddr_in to;
4. hp = gethostbyname(argv[1]);
5. …
6. /* relleno de la estructura para la direccion destino */
7. memset((char *)&to, (unsigned char) 0, sizeof(to));
8. memcpy((char *)&to.sin_addr, hp->h_addr, hp->h_length);
9. to.sin_family = hp->h_addrtype;
10. to.sin_port = htons(puerto);
11. …
12. r = connect(s, (struct sockaddr *)&to, sizeof(to));
```

Apaga la advertencia del compilador



Comunicación en modo conectado (1)

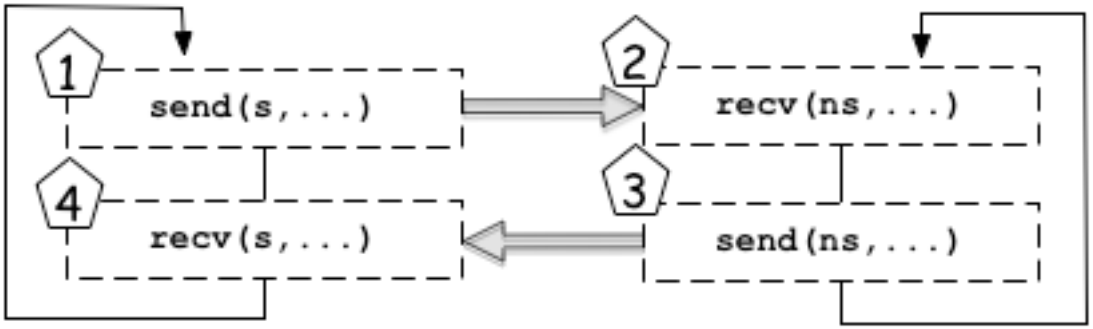
- En Linux o Unix se utilizan las llamadas a sistema read() y write().
- En realidad son hechas a semejanza de las funciones para escribir en archivos.
- En la secuencia que se presenta a continuación, 1 y 4 ocurren en el cliente, 2 y 3 ocurren en el servidor.




```
ssize_t read(int descriptor, void *buf, size_t tamano)
ssize_t write(int descriptor, const void *buf, size_t tamano)
```

Comunicación en modo conectado (2)

- En Windows las funciones que se utilizan para el intercambio de mensajes se llaman send() y recv().
- Son iguales que write y read solo que tienen una bandera de más. Valores posibles: MSG_OOB, MSG_PEEK.




```
ssize_t recv(int descriptor, void *buf, size_t tamano, int flags)
ssize_t send(int descriptor, const void *buf, size_t tamano, int flags)
```

Cierre de conexiones.

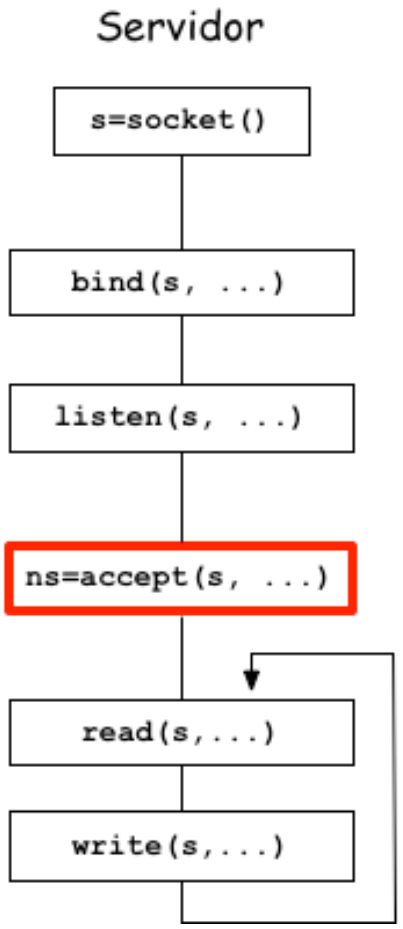
- Función **`close()`**
  - Cierra la conexión especificada en el descriptor (un socket por ejemplo).
  - El socket no se cierra sino hasta que todos los procesos dependientes cierran.
- Función **`shutdown()`**

```
int shutdown(int s, int how)
```

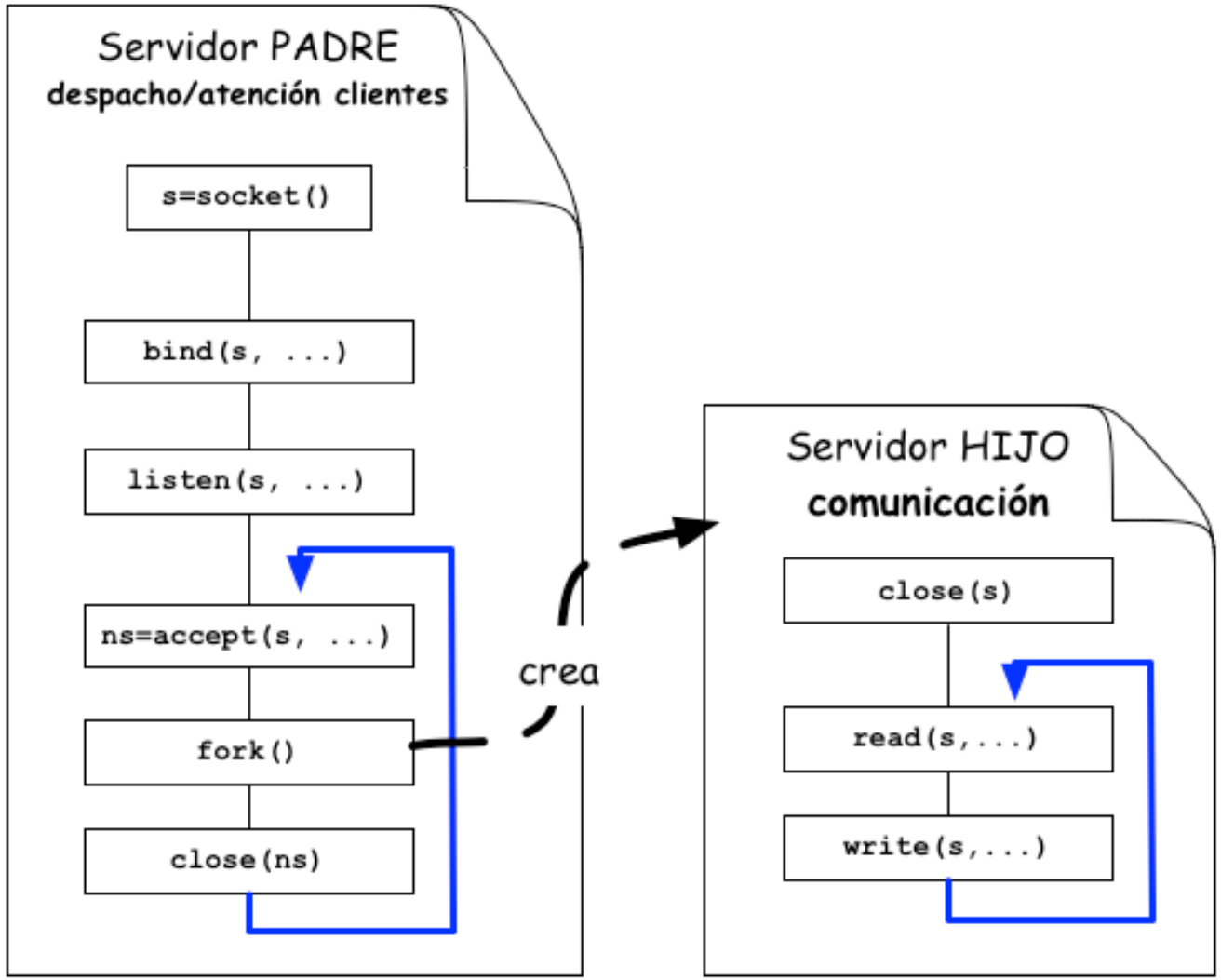
  - Parámetro how:
    - SHUT_RD: cerrar solo para lectura
    - SHUT_WR: cerrar solo para escritura
    - SHUT_RDWR: cerrar lectura y escritura

¿Cómo puede hacerse que un servidor sea concurrente?

- Problemas
  - La función `accept()` es **bloqueante** (en todo el sentido).
  - Además, `accept()` recibe 1 sola conexión
  - Lo ideal sería que el servidor pudiera duplicarse y luego del `accept()`: 1) seguir esperando otras conexiones, 2) tratar la comunicación.
- Soluciones
  - Generar un nuevo proceso (duplicado al servidor)
  - Utilizar procesos ligeros llamados 'threads'



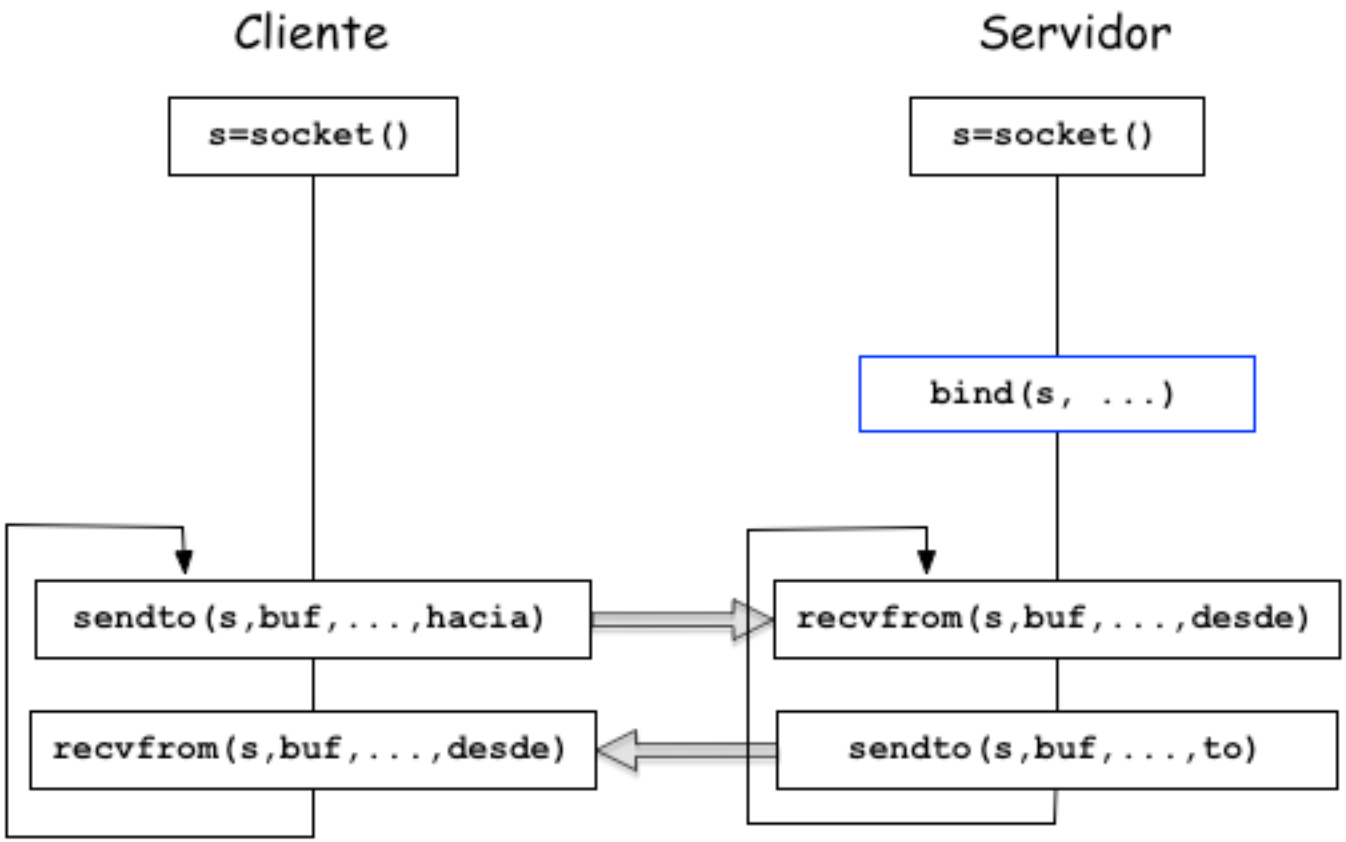
Modelo de Servidor Concurrente.



Ejemplo creación de proceso hijo.

```
1.  int pid;
2.   …
3.   pid = fork();
4.   if pid == 0 {  // estamos en  el proceso hijo
5.      // establecer la comunicación
6.   } else if pid > 0 { // estamos en el proceso padre
7.      // seguir escuchando comunicaciones entrantes
8.   } else if pid < 0 {
9.      // lanzar mensaje de error
10.  }
```

Comunicación no confiable (sockets sin conexión).



Funciones sendto() y recvfrom()

```
ssize_t  sendto(int socket,
                const void *buf,
                size_t long_buf,
                int opciones,
                const struct sockaddr * hacia,
                socklen_t long_hacia);

ssize_t  recvfrom(int socket,
                 void *buf,
                 size_t long_buf,
                 int opciones,
                 struct sockaddr * desde,
                 socklen_t * long_desde);
```

Otras funciones…

- Obtener el nombre o la dirección de una máquina.
  - `struct hostent * gethostbyname(…)`
    - Mediante ésta función se obtiene una estructura que sirve a identificar una máquina. Basta con pasarle el nombre o la dirección IP (ej: a.b.c.d) para obtener una respuesta.
  - `struct hostent * gethostbyaddr(…)`
    - Devuelve la misma estructura que la anterior solo que la información de entrada (ej: dirección IP) se da bajo la forma de una estructura (sockaddr_in.sin_addr)

Funciones gethostbyname() y gethostbyaddr()

```
1.  #include <netdb.h>
2.  struct hostent *gethostbyname(const char *name);
3.  struct hostent *gethostbyaddr(const char *addr,
4.                                   int longitud,
5.                                   int tipo);
6.  struct hostent {
7.      char   *h_name; // nombre oficial del host
8.      char  **h_aliases; // lista de los alias
9.      int   h_addrtype; // tipo de la direccion del host
10.     int   h_length;   // longitud de la direccion
11.     char  ** h_addr_list; // lista de direcciones
12. }
```

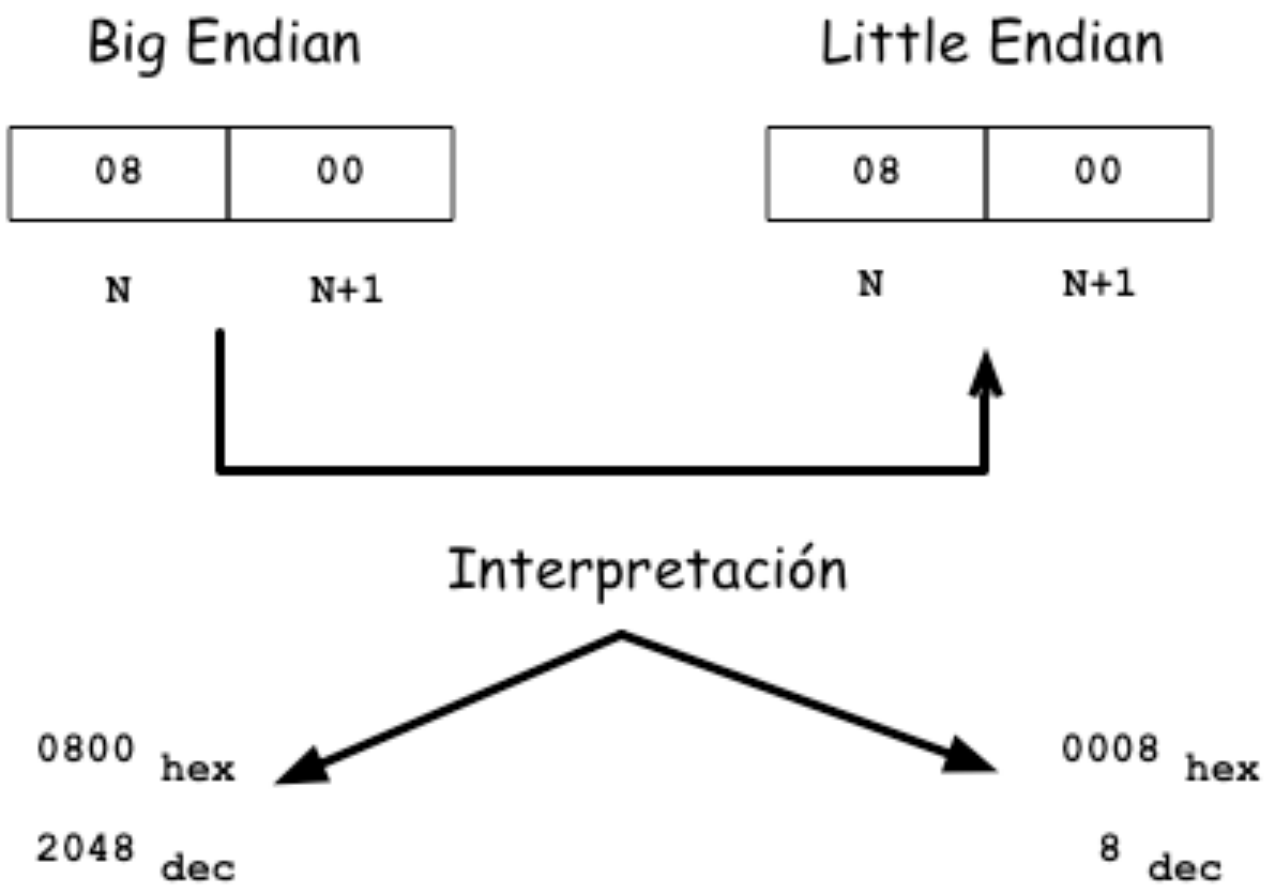
Servicios para la resolución de nombres

- Las funciones vistas anteriormente son la interfaz en C del sistema de búsqueda de nombres para un host (situado dentro del OS).
  - Este procedimiento se encuentra bajo el control de un archivo llamado /etc/host.conf o /etc/nsswitch.conf.
    - Archivo /etc/hosts
    - Servicio NIS
    - Servicio DNS
- La tarea de resolución de nombres no le compete al programador sino al administrador del sistema.

Uso de big endian vs. little endian.

- Se trata del problema de representación de la información en la máquina.
  - Por ejemplo $2^{11} = 2048$, y se representa 0800 en hexadecimal (0x0800 en C).
  - En memoria se representa así para Big Endian:
    - Dir N [08], N+1 [00] { motorola (ppc), sun (sparc) }
  - Para Little Endian:
    - Dir N [00], N+1 [08] { Intel }

El problema LE vs BE



Solución al problema LEBE

- Hay que respetar una convensión máquina/red y red/máquina.
  - La red es Big Endian
    - Máquina a Red → usar htons, htonl (short y long)
    - Red a Máquina → usar ntohs, ntohl (short y long)
  - Ejemplo:
    - sin.sin_port = htons(2334);

Opciones en Sockets

- Las funciones manejadores de opciones en sockets son:

1. `int getsockopt(int s, int level, int optname, void * optval, socken_t *optlen);`
2. `int setsockopt(int s, int level, int optname, const void *optval, socklen_t optlen);`

s : descriptor del socket

level : que alcance tiene la operación SOL_SOCKET, SOL_IP, SOL_TCP

optname : nombre de la opción

optval : valor de la opción

optlen : longitud de la opción

Las opciones de los sockets

- SO_BROADCAST: Permite una difusión en broadcast en un socket UDP.
- SO_REUSEADDR: Puede reutilizarse la misma dirección dada a bind()
- SO_KEEPALIVE: Provoca el envío de un mensaje para constatar que el servidor está vivo cuando se esta en modo conectado (TCP) y la conexión ha estado inactiva por algún tiempo.
- SO_RCVBUF, SO_SNDBUF: tamaño de los buffers de envío y recepción
- SO_LINGER: controla el envío de datos en el momento del cierre de la conexión.
- Más opciones en **man 7**.

Para finalizar la clase, realice la práctica de sockets que será enviada por correo.