



# **Clase 5**

## **Networking**



## **Sockets**

- **Conector a canal de comunicaciones bidireccional.**
- **Permiten comunicar procesos**
- **Usan un file descriptor**



## **Sockets**

- **Internet sockets**
  - Manejan direcciones de red
  - Comunicación de sistemas distribuidos
  - Comunicación local
  - Multiplataforma
- **Unix sockets**
  - Comunicación local
  - Solo sistemas unix-like

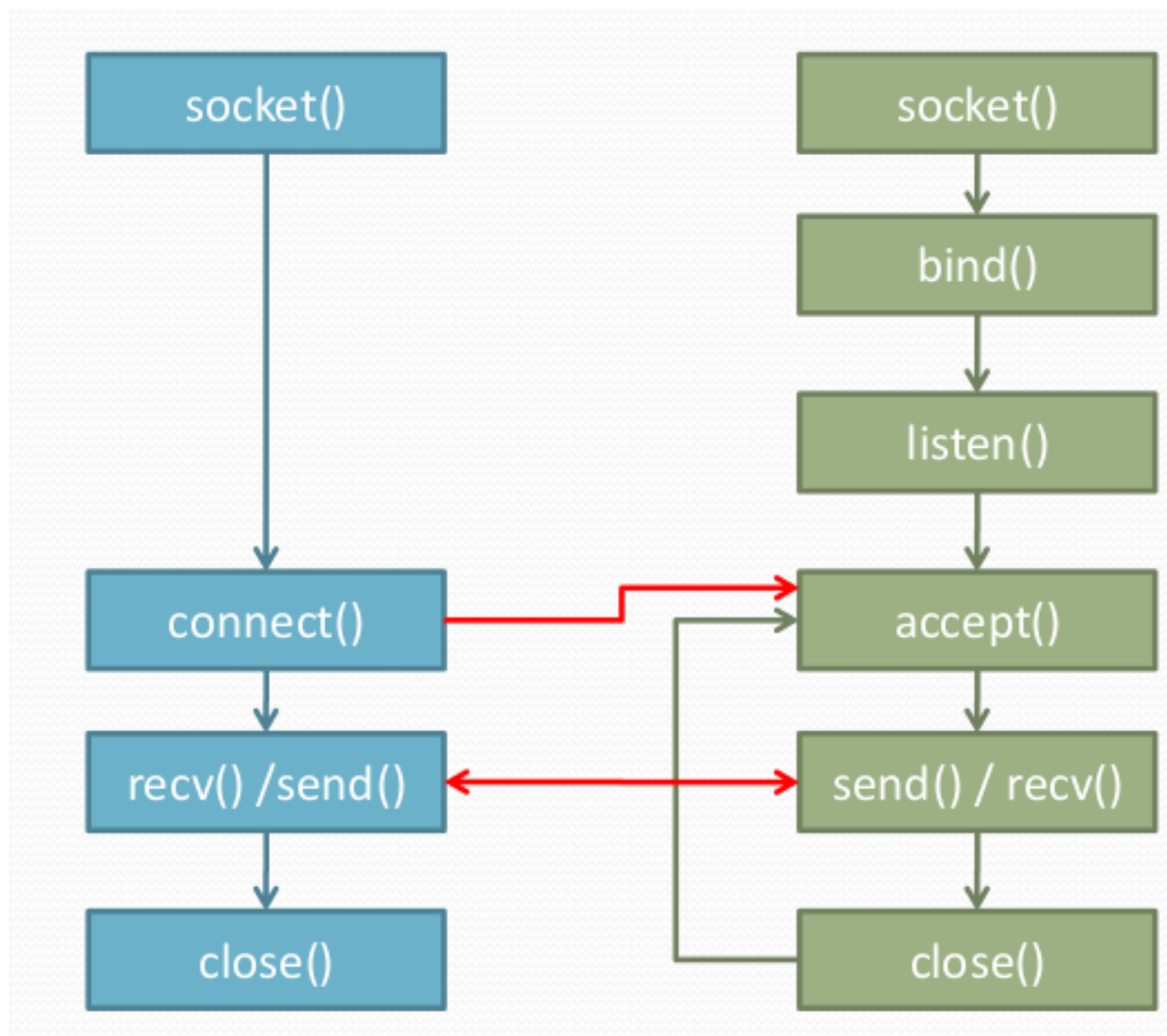


## **Internet Sockets**

- **Stream sockets**
  - SOCK\_STREAM
  - Orientado a conexión
  - Sin límite de tamaño de datos
  - TCP
- **Datagram sockets**
  - SOCK\_DGRAM
  - No orientado a conexión
  - Limite en tamaño de packet
  - UDP
- **Raw sockets**
  - Sobre capa IP

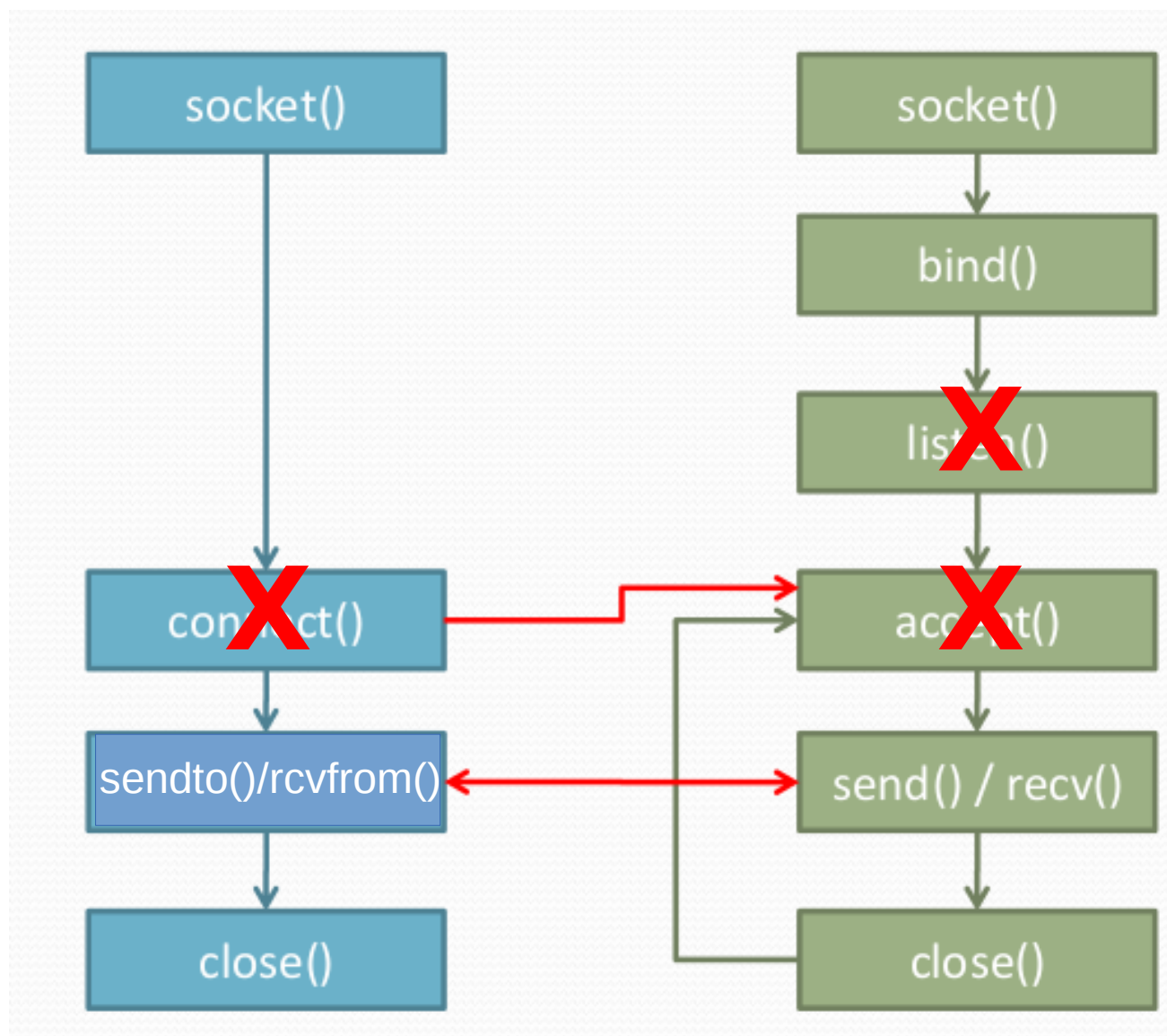


## API para Stream sockets





## API para Dgram sockets





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

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

- La función crea un socket “desconectado” en el dominio de comunicación especificado.
- Devuelve un File Descriptor
- **domain:** Familia de protocolos P. ej. AF\_INET (ipv4) o AF\_INET6 (ipv6) o AF\_UNIX (local), etc.
- **type:** Tipo de socket P. ej. SOCK\_STREAM (tcp) o SOCK\_DGRAM (udp) o SOCK\_RAW
- **protocol:** Normalmente 0



## Nodo A



Internet

Creacion

socket()

socket

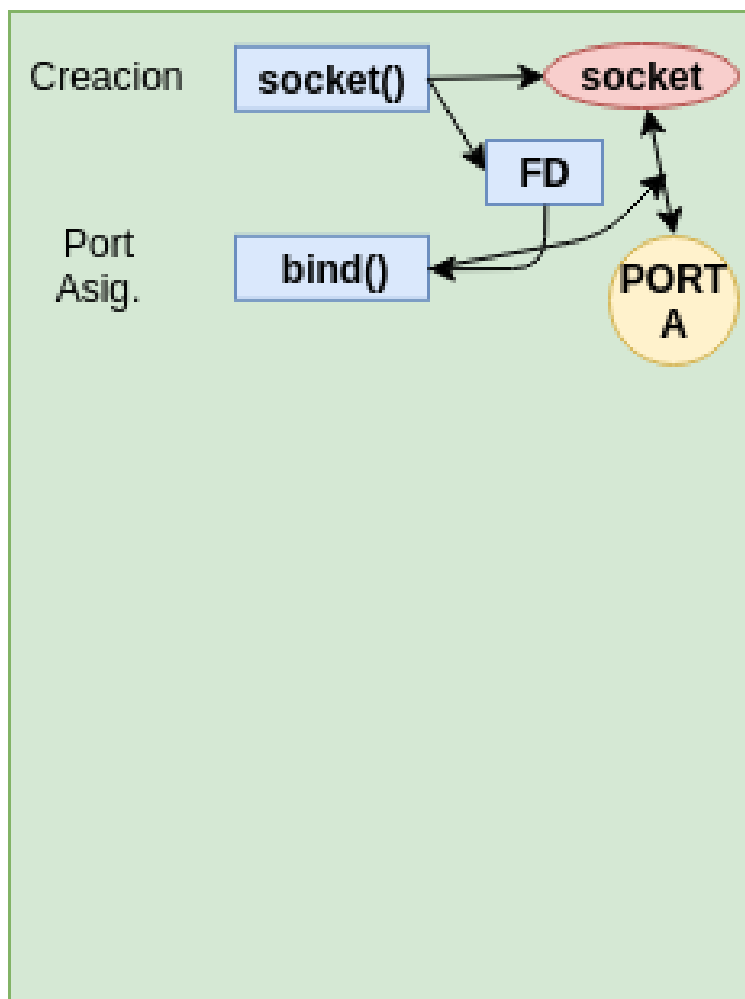
FD





Internet

## Nodo A





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

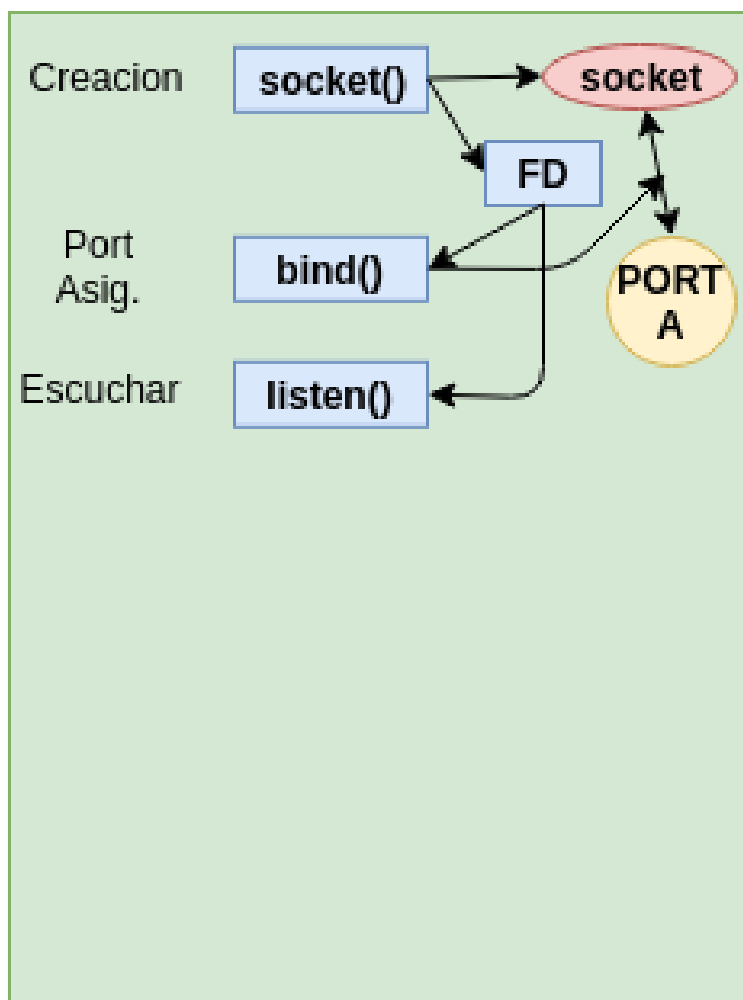
```
int bind(int socket, const struct sockaddr *address,  
         socklen_t address_len);
```

- Asigna una dirección (por ej. IP:Puerto) al socket.
  - **socket**: FD generado por socket()
  - **address**: estructura con los campos ip y puerto
  - **address\_len**: Size en bytes de la struct anterior
- sockaddr es un tipo “padre” para los datos particulares de cada protocolo que en realidad se deben usar. por ej.:
  - **sockaddr\_in** para ipv4
  - **sockaddr\_in6** para ipv6



Internet

## Nodo A





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

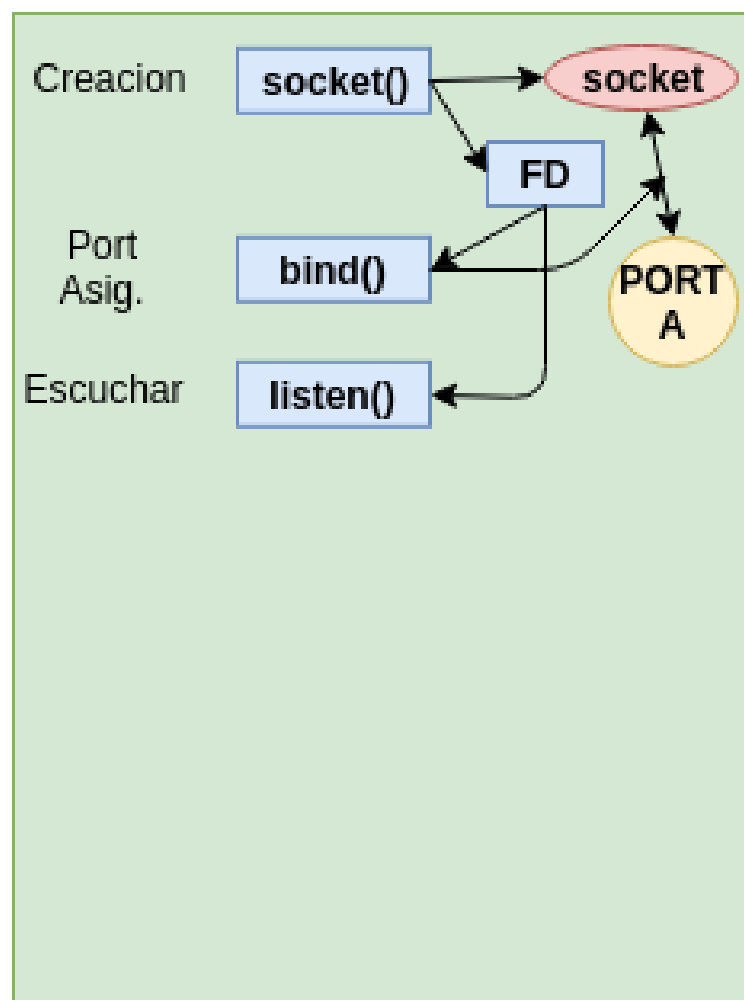
```
int listen(int socket, int backlog);
```

- Al crear un socket el mismo se encuentra en estado “activo” (listo para realizar un connect())
- La función marca al socket como “pasivo” (o en estado listening) para que pueda ser usado para escuchar conexiones entrantes y aceptarlas con la función accept().
- **socket**: FD generado por socket()
- **backlog**: Cantidad de pedidos de conexión que se almacenarán mientras se responde al pedido de conexión en curso de ser aceptado.

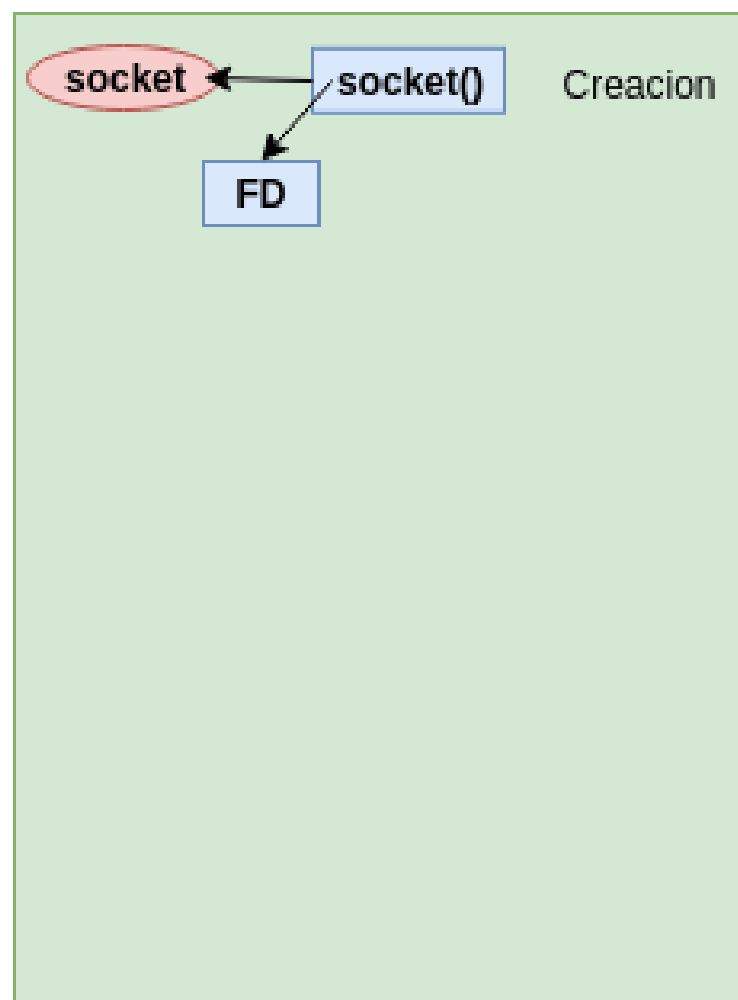


Internet

## Nodo A



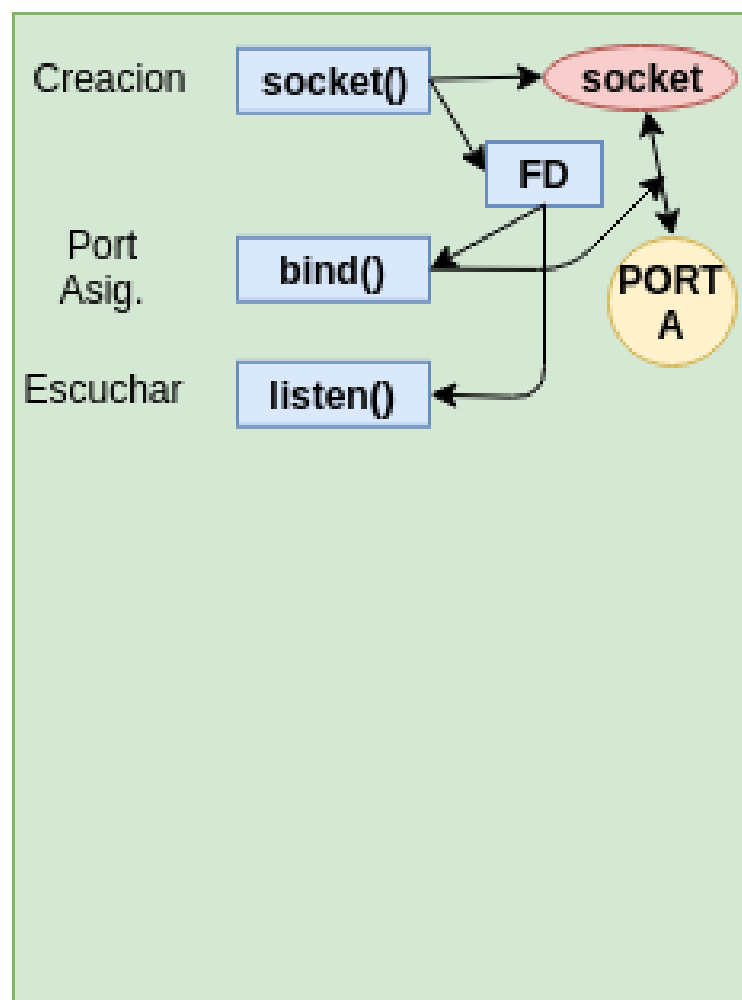
## Nodo B



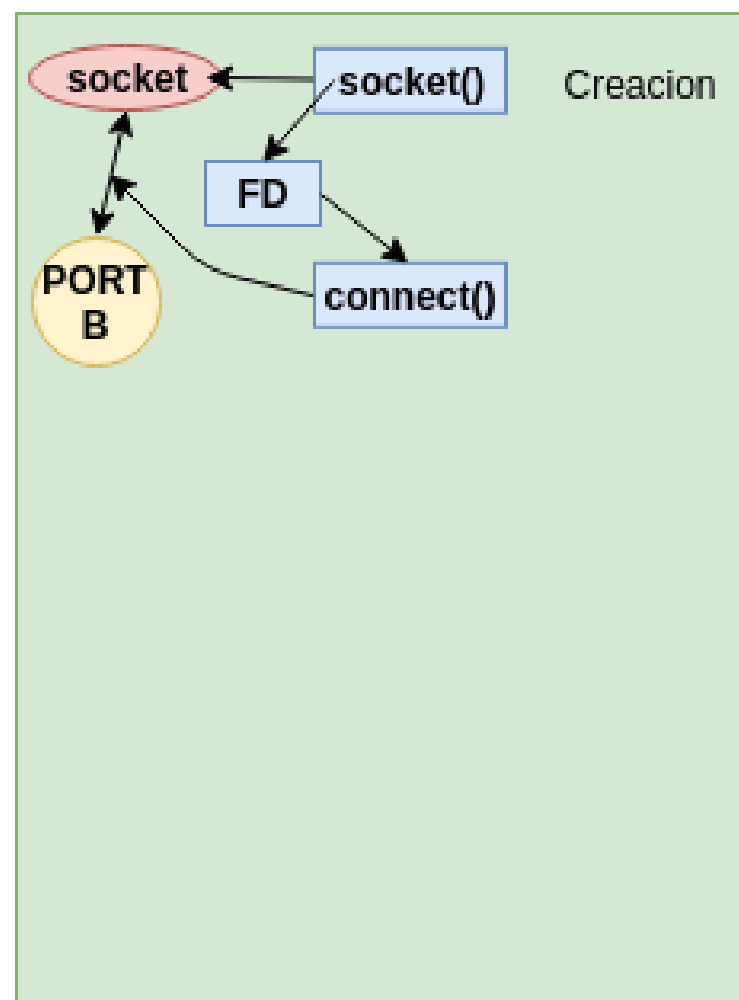


Internet

## Nodo A



## Nodo B





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

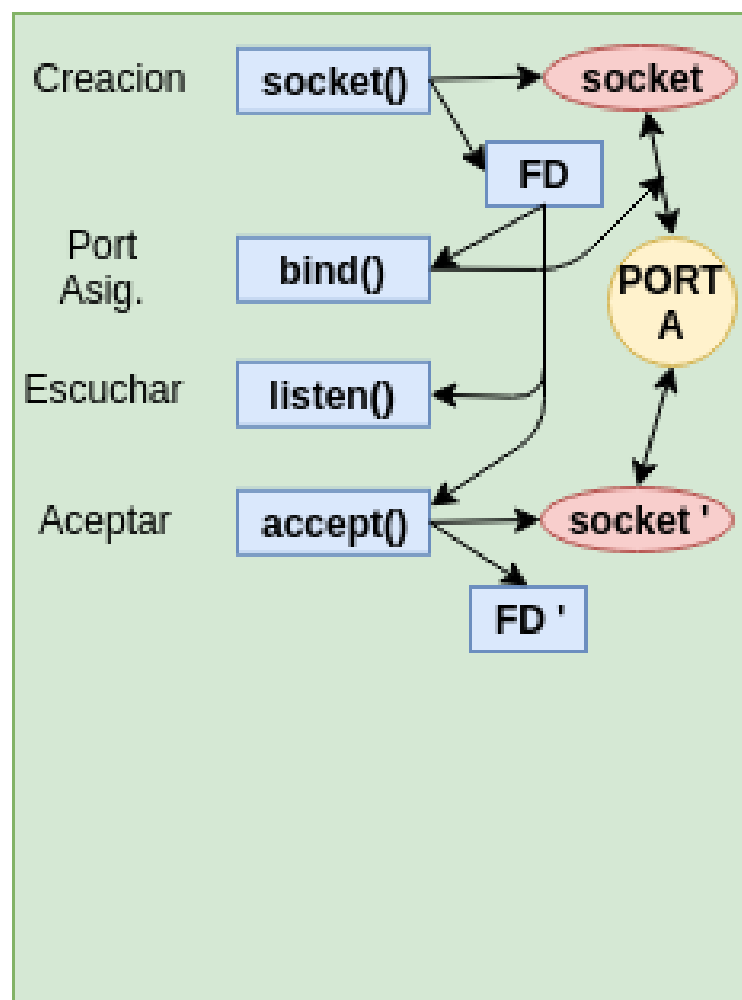
```
int connect(int socket,  
            const struct sockaddr *address,  
            socklen_t address_len);
```

- Trata de hacer una conexión sobre un socket orientado a conexión o setea los parámetros de conexión sobre un socket no orientado a conexión.
- **socket**: FD generado por `socket()`
- **address**: Par ip puerto destino
- **address\_len**: Size en bytes de la struct anterior

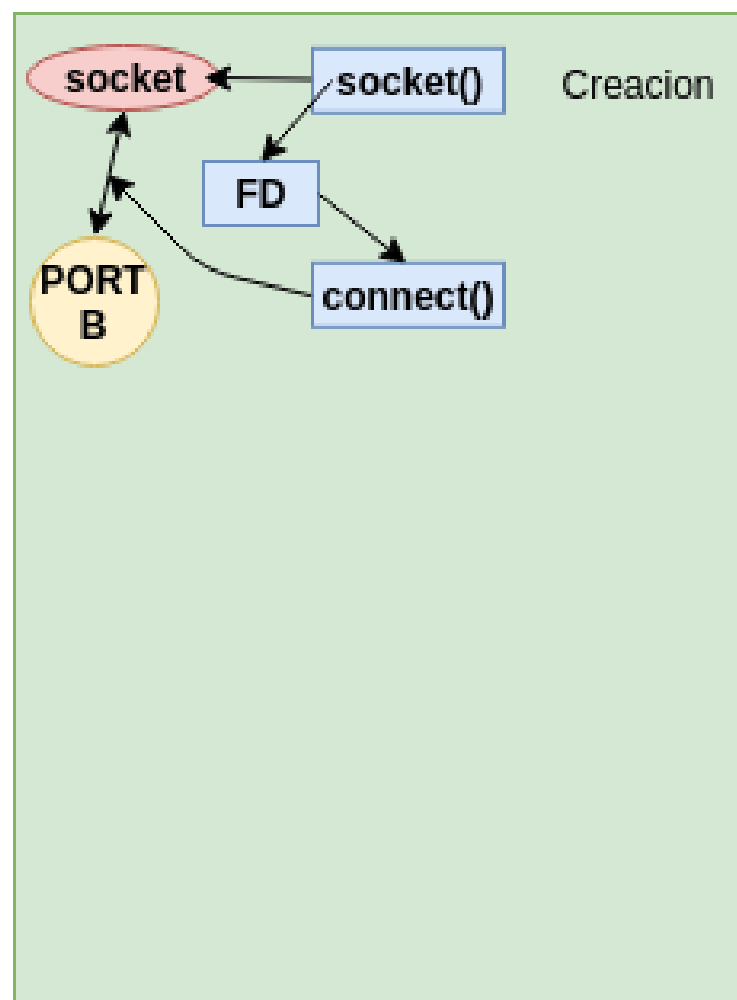


Internet

## Nodo A



## Nodo B







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

```
int accept(int socket,  
           struct sockaddr* address,  
           socklen_t* address_len);
```

- Extrae la primera conexión de la cola de conexiones pendientes.
- Crea un nuevo socket con el mismo tipo, protocolo y familia de direcciones que el original.
- Crea un FD para el nuevo socket.



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

```
int accept(int socket,  
           struct sockaddr* address,  
           socklen_t* address_len);
```

- **socket:** FD generado por `socket()` (el que está en estado "listening")
- **address:** Acá se escribirán los datos del socket remoto que se conectó (ip y puerto).
- **address\_len:** Es un "value-result argument"
  - Se le pasa el size en bytes de la struct anterior.
  - Devuelve el size que se escribió en dicha struct.
- Devuelve el FD del nuevo socket generado.

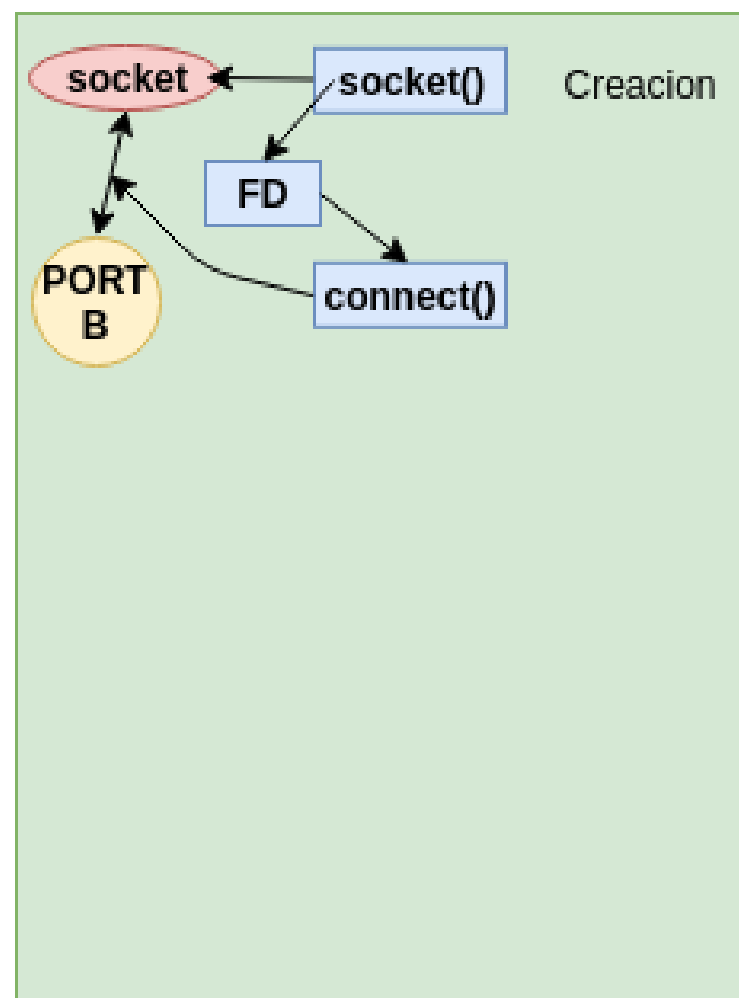
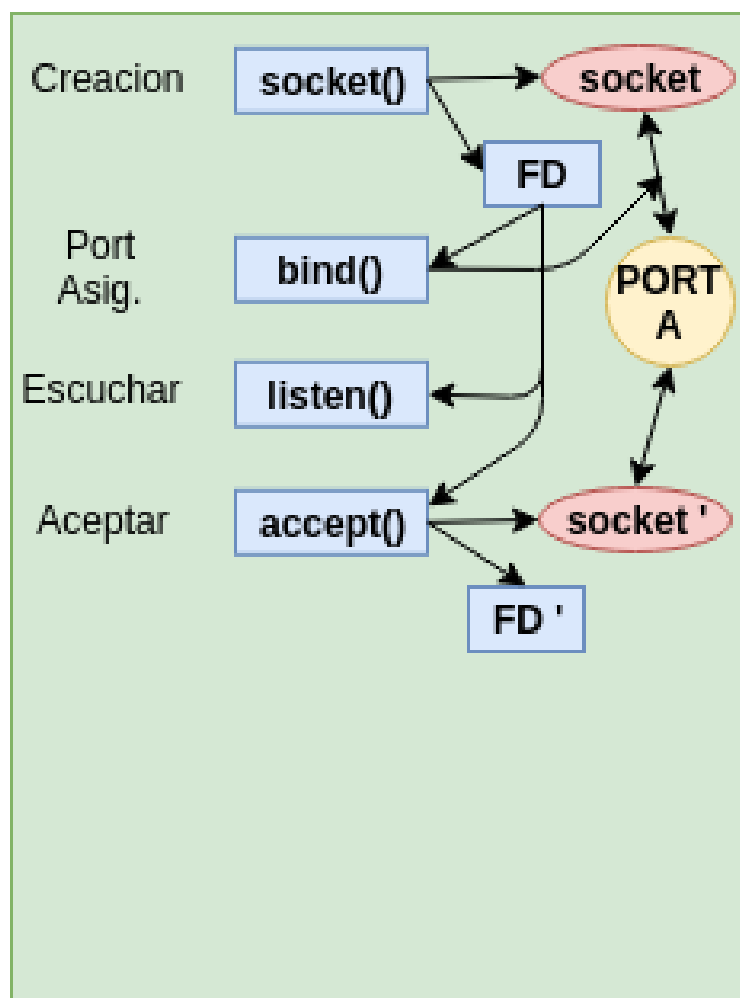


## ¿Por qué se genera otro socket+FD?

Nodo A

Internet

Nodo B





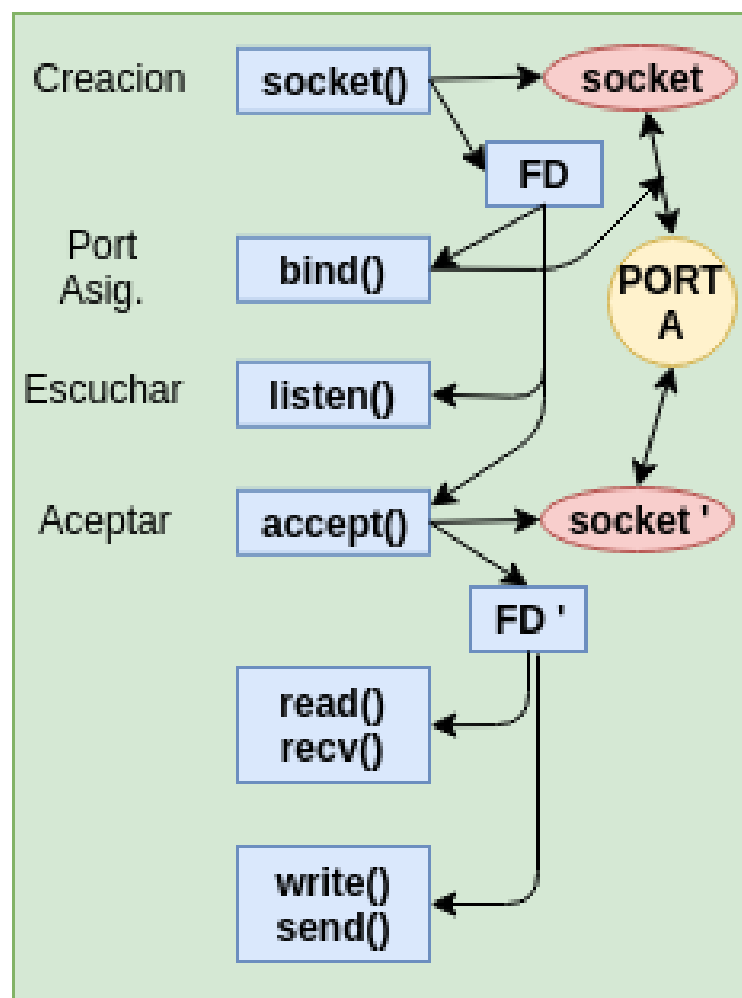
## ¿Por qué se genera otro socket+FD?

- El socket generado es “activo” no está escuchando conexiones entrantes.
- Se utilizará para enviar y recibir datos mediante el canal establecido
- Cada socket posee 4 datos:
  - IP server
  - Port server
  - IP client
  - Port client
- El socket generado estará “atado” al client que estableció la conexión y permitirá la comunicación con el mismo.

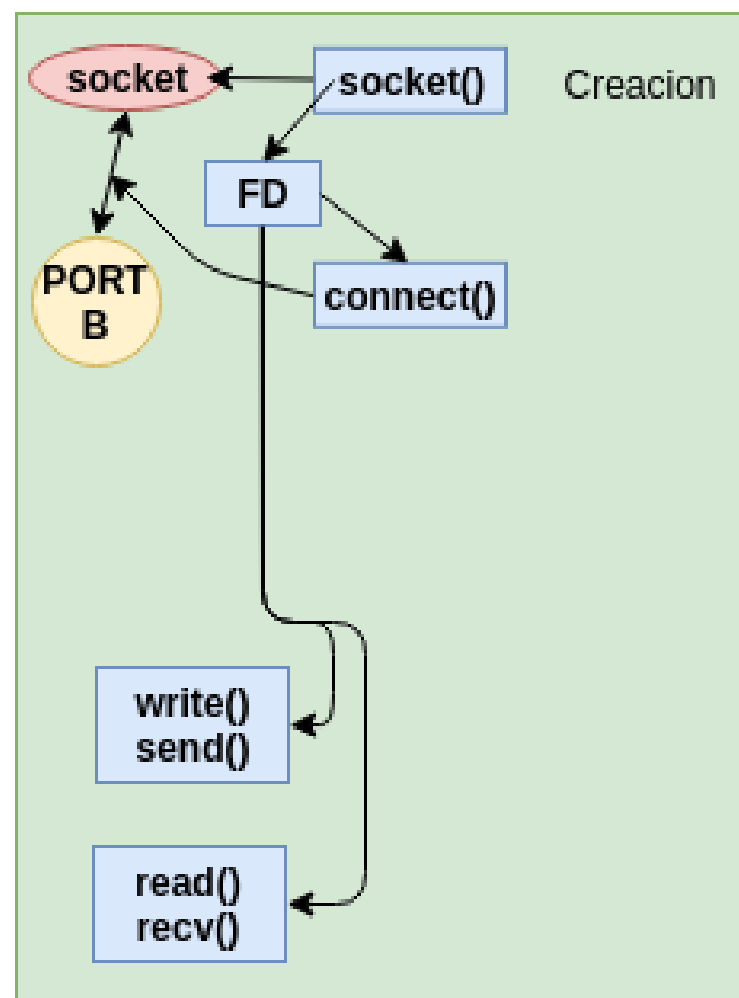


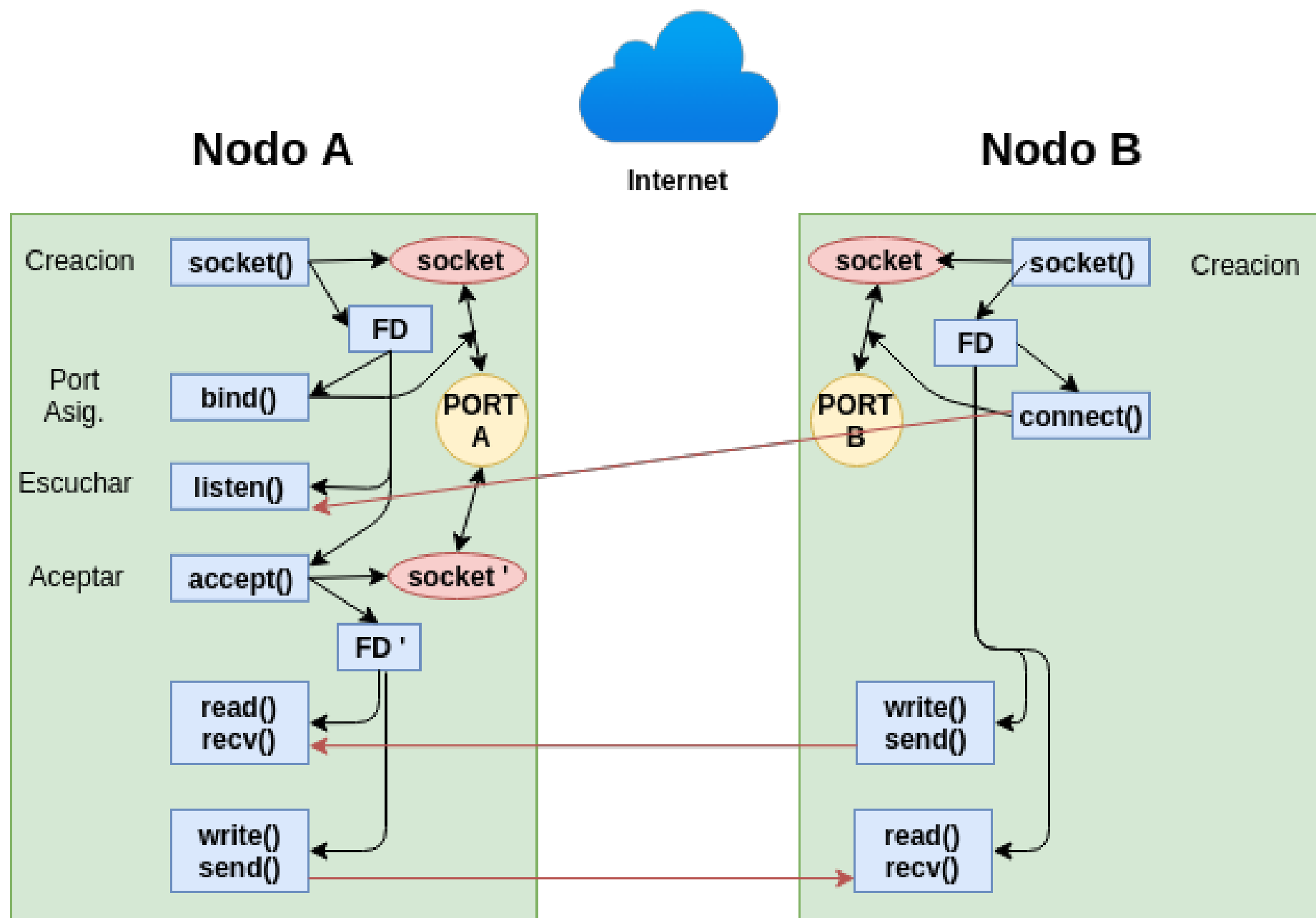
Internet

## Nodo A



## Nodo B







## **Ejemplos TCP y UDP**



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

```
ssize_t send(int socket, const void *buffer,  
             size_t length, int flags);
```

- **socket**: FD asociado al socket.
  - **buffer**: Puntero a array con bytes a transmitir.
  - **length**: Cantidad de bytes a transmitir.
  - **flags**: Tipo de transmisión (ver documentación)
- 
- Solo enviará el msg si el socket está conectado.
  - Devuelve el número de bytes enviados o -1.





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

```
ssize_t recv(int socket, void *buffer,  
             size_t length, int flags);
```

- **socket**: FD asociado al socket.
- **buffer**: Puntero a array con bytes donde quedará el msg recibido.
- **length**: Cantidad de bytes max a recibir.
- **flags**: Define comportamiento (No bloqueante, etc.)
  
- Solo recibirá el msg si el socket está conectado.
- Devuelve el número de bytes recibidos o -1.



- No orientado a conexión: Usamos **recvfrom** y **sendto**

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

```
ssize_t recvfrom(int socket, void* buffer,  
                 size_t length, int flags,  
                 struct sockaddr* address,  
                 socklen_t* address_len);
```

```
ssize_t sendto(int socket, const void* message,  
               size_t length, int flags,  
               const struct sockaddr* dest_addr,  
               socklen_t dest_len);
```



- Un socket puede cerrarse al ejecutar `close()` o porque el proceso con el socket asociado terminó.
- Si desde el otro lado de la conexión se ejecuta `read()` o `recv()`, se recibirá **EOF**.
- Si desde el otro lado de la conexión se ejecuta `write()` o `send()`, se recibirá la signal **SIGPIPE** y la syscall devolverá un error (**EPIPE**). (Esto puede no ocurrir en el 1er envío).
  - Solución: Ignorar la signal y analizar el error.
  - Solución: Usar `send()` con flag `MSG_NOSIGNAL`.

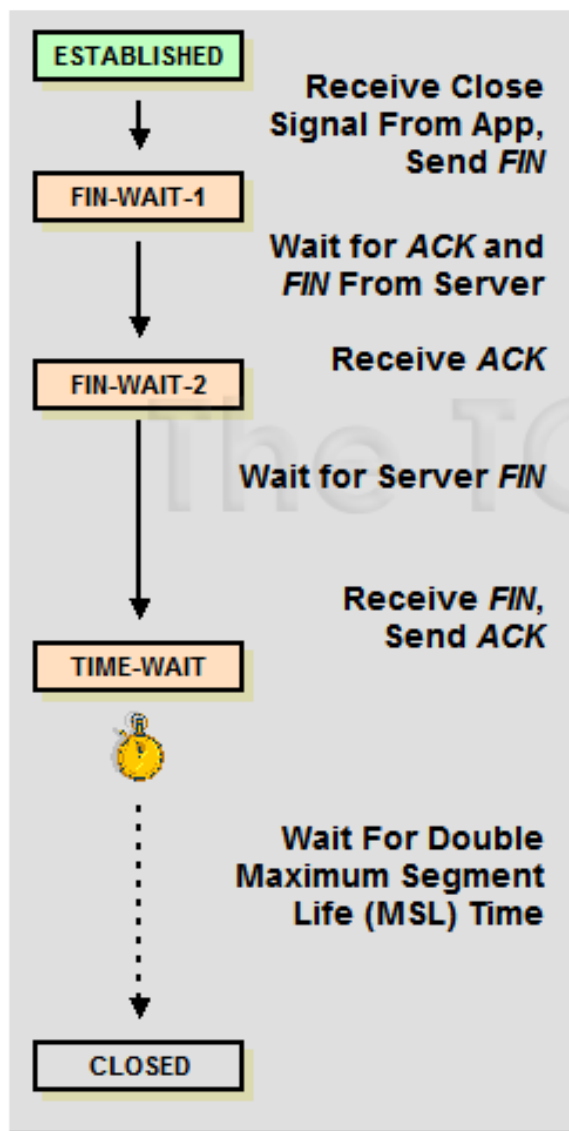
De lo contrario al recibir la signal el proceso termina.



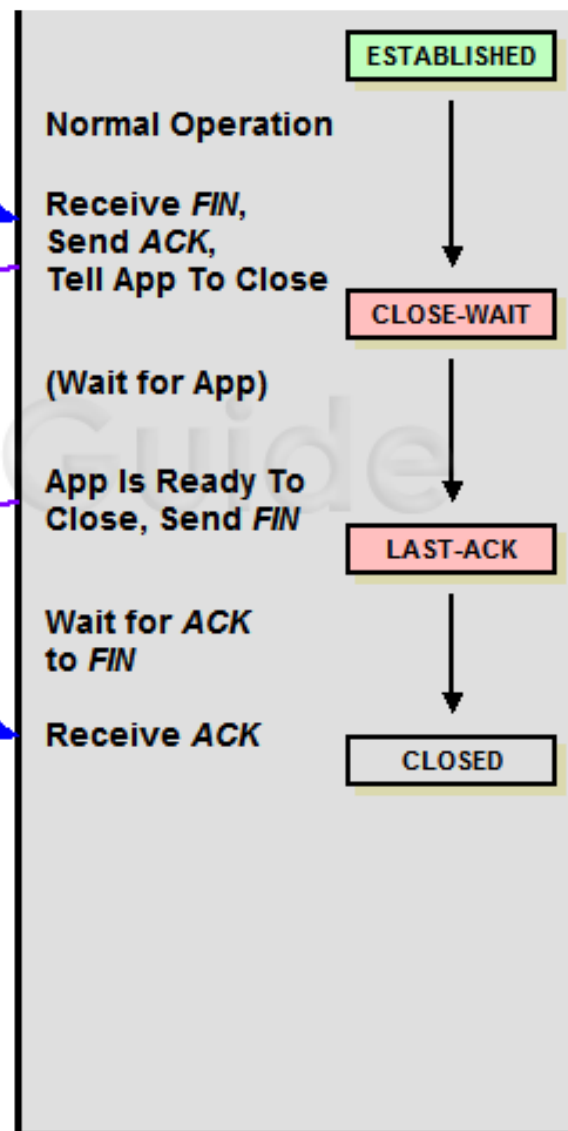
- Si el proceso server cierra con una conexión establecida, el puerto quedará ocupado para el sistema operativo.
- Al volver a ejecutar el proceso, la syscall `bind()` dará el error "Puerto en uso".
- ¿Por qué ocurre esto?



## Active close



## Passive close



- **Active close:** Acción de cerrar la conexión siendo el que comienza el proceso.
- **Passive close:** A quien le cierran la conexión.
- Supongamos una conexión establecida:
  - 1) El server hace un close del socket (Active close)
  - 2) El server envía FIN al client, quien contesta ACK y queda en estado CLOSE\_WAIT.
  - 3) El server queda en estado FIN\_WAIT2
  - 4) El cliente hace un close del socket.
  - 5) El cliente envía FIN al server, quien contesta ACK y queda en estado TIME\_WAIT.
  - 6) El cliente queda en estado CLOSED.
- El estado TIME\_WAIT puede durar algunos minutos.



- Mientras esté en este estado, `bind()` dará error de "Puerto ocupado".
- No importa si el proceso termina o no.
- **Conclusion:** Siempre el que hace el `active close` (el que cierra primero), queda en `TIME_WAIT`.
- ¿Por qué se necesita `TIME_WAIT`?





- **¿Por qué se necesita TIME\_WAIT?**
- Permite descartar segmentos retrasados que llegan, que pueden ser de una conexión anterior.
- Si el último ACK (el del server) no le llega al cliente, el cliente va a reenviar el FIN: Si el server hubiera pasado a CLOSED, en vez de quedarse en TIME\_WAIT, el cliente no recibiría el ACK y creería que la conexión terminó con un error en vez de creer que terminó normalmente.





- **¿Cómo evitarlo?**

- Si el cliente termina primero.
- Si el cliente hace un envío con la conexión cerrada, enviará un RST. Lo cual quita al server de TIME\_WAIT.
- Setear el flag SO\_REUSEADDR en el socket, esto hace que el puerto esté disponible aunque el estado sea TIME\_WAIT (no recomendado)



## Funciones útiles: htons() y inet\_pton()

```
bzero((char *) &serveraddr, sizeof(serveraddr));  
serveraddr.sin_family = AF_INET;  
serveraddr.sin_port = htons(4096);  
inet_pton(AF_INET, "127.0.0.1",  
          &(serveraddr.sin_addr));
```

- **htons():**

- Recibe el número de puerto y lo devuelve en el formato para el campo sin\_port de la struct sockaddr.

- **inet\_pton():**

- Recibe la IP (v4 o v6) en formato texto y la devuelve en el formato para el campo sin\_addr de la struct sockaddr.



## Funciones útiles: `inet_ntop()` y `ntohs()`

```
inet_ntop(AF_INET, &(clientaddr.sin_addr),  
          ipClient, sizeof(ipClient));
```

```
int port = ntohs(clientaddr.sin_port);
```

```
printf(" desde ip:%s port:%d", ipClient, port);
```

- **`inet_ntop()`:**

- Nos devuelve en formato texto la IP. (IP v4 o v6)

- **`ntohs()`:**

- Se le pasa un campo `sin_port` de una struct `sockaddr_in`
- Nos devuelve el número de puerto.



## Funciones útiles: getaddrinfo() y freeaddrinfo()

```
struct addrinfo hints;  
struct addrinfo* result;  
  
memset(&hints, 0, sizeof(struct addrinfo));  
hints.ai_family = AF_INET; // ipv4  
hints.ai_socktype = SOCK_STREAM; // tcp  
hints.ai_flags = AI_PASSIVE; // Para usar con accept  
  
int r = getaddrinfo(NULL, "4096", &hints, &result);  
                                // NULL para localhost  
  
if (r != 0)  
{  
    fprintf(stderr, "getaddrinfo: %s", gai_strerror(r));  
}
```

- “result” es una lista enlazada. Usamos la 1er posición.



## Funciones útiles: `getaddrinfo()` y `freeaddrinfo()`

```
if (bind(s, (struct sockaddr*)result->ai_addr,  
          result->ai_addrlen) == -1) {  
    close(s);  
    perror("listener: bind");  
}
```

```
freeaddrinfo(result);
```

- “result” es una lista enlazada. Usamos la 1er posición para hacer el `bind()`.
- Los datos de IP y puerto estarán cargados, a partir del `hostname` y `port` que le dimos a `getaddrinfo()`.
- Deberemos liberar la lista con `freeaddrinfo()`.



## **Bibliografía**

- Brian “Beej Jorgensen” Hall. (2015). Beej's Guide to Network programming.
- Michael Kerrisk. (2010). The Linux programming interface. No Starch Press, Inc.
- The Open Group Base Specifications Issue 7, 2018 edition  
<http://pubs.opengroup.org/onlinepubs/9699919799>
- TCP Guide, <http://www.tcpipguide.com>
- Alejandro Furfaro (2016). Presentación Internetworking.