Sockets TCP/IP en C - Sockets

Di Paola Martín
martinp.dipaola <at> gmail.com

Facultad de Ingeniería
Universidad de Buenos Aires

De qué va esto?

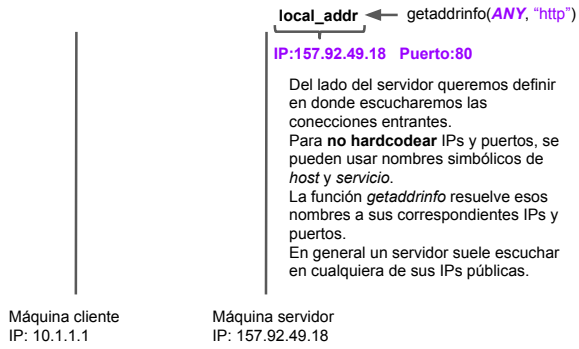
Resolución de nombres

Canal de comunicación TCP

- Establecimiento de un canal
- Envio y recepción de datos
- Finalización de un canal

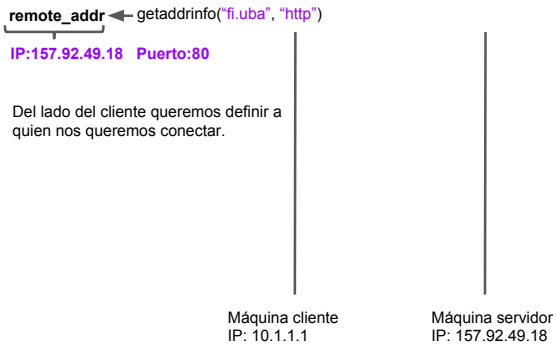
Resolución de nombres

Resolución de nombres: desde donde quiero escuchar



- El servidor tiene que definir desde donde quiere recibir las conexiones.
- Hay más esquemas posibles pero solo nos interesa definir la IP y el puerto del servidor.
- Sin embargo, hardcodear la IP y/o el puerto es una mala práctica. Mejor es usar nombres simbólicos: host name y service name.
- La función **`getaddrinfo`** se encargara de resolver esos nombres y llevarlos a IPs y puertos.

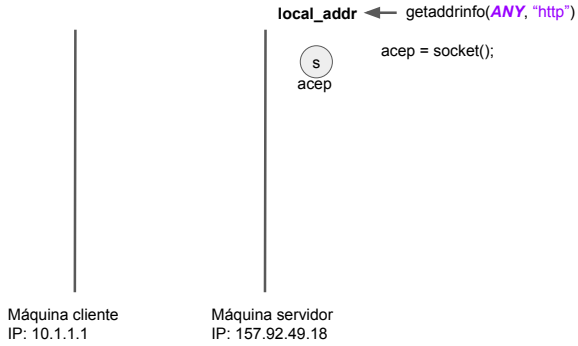
Resolución de nombres: a quien me quiero conectar



Canal de comunicación TCP

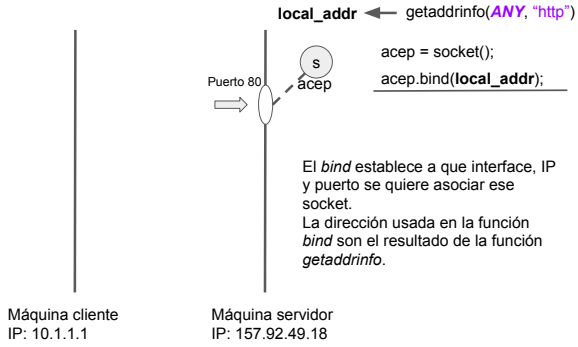
Establecimiento de un canal

Creación de un socket



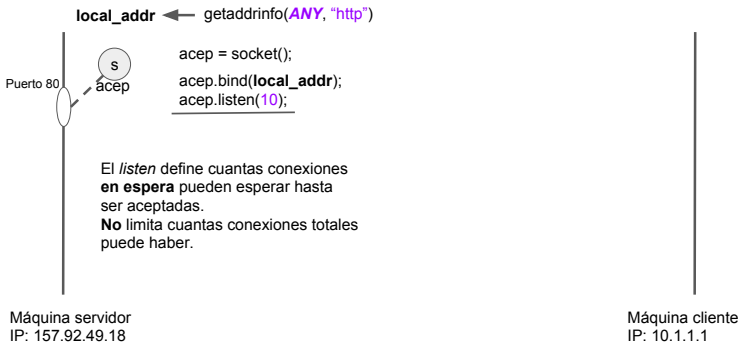
- Crear un socket no es nada mas que crear un file descriptor al igual que cuando abrimos un archivo.

Enlazado de un socket a una dirección



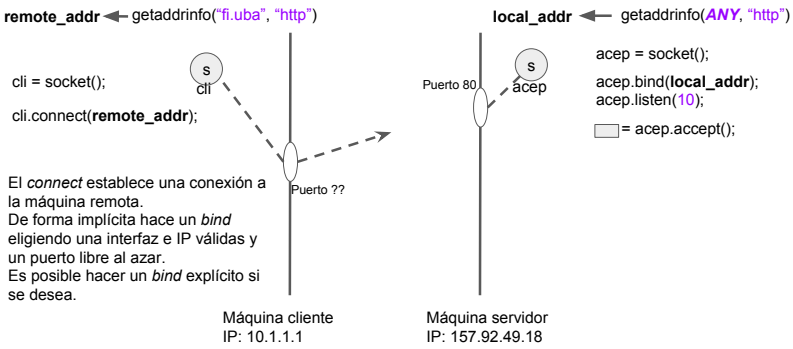
- A los sockets se los puede enlazar o atar a una dirección IP y puerto local para que el sistema operativo sepa desde donde puede enviar y recibir conexiones y mensajes.
- El uso mas típico de **`bind`** se da del lado del servidor cuando este dice "quiero escuchar conexiones desde mi IP pública y en este puerto".
- Sin embargo el cliente también puede hacer **`bind`** por razones un poco mas esotéricas.

Socket aceptador o pasivo



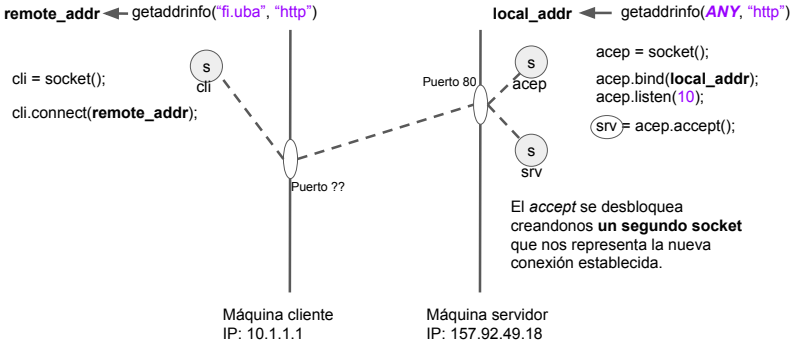
- Una vez enlazado le decimos al sistema operativo que queremos escuchar conexiones en esa IP/puerto.
- La función **`listen`** define hasta cuantas conexiones en "espera de ser aceptadas" el sistema operativo puede guardar.
- La función **`listen`** NO define un límite de las conexiones totales (en espera + las que estan ya aceptadas). No confundir!
- Ahora el servidor puede esperar a que alguien quiera conectarse y aceptar la conexión con la función **`accept`**.
- La función **`accept`** es bloqueante.

Conexión con el servidor: estableciendo conexión



- El cliente usa su socket para conectarse al servidor. La operación **`connect`** es bloqueante.

Conexión con el servidor: aceptando la conexión

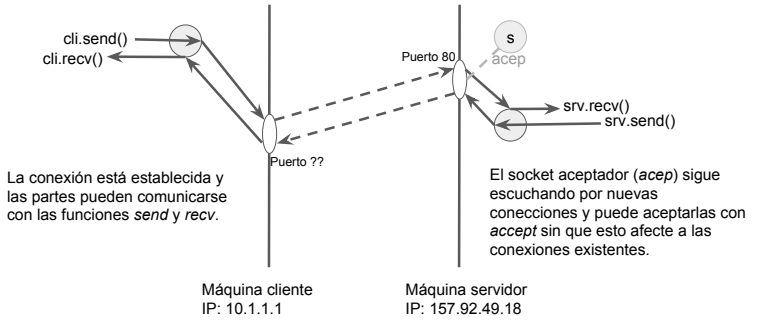


- La conexión es aceptada por el servidor: la función **`accept`** se desbloquea y retorna un nuevo socket que representa a la nueva conexión.

Canal de comunicación TCP

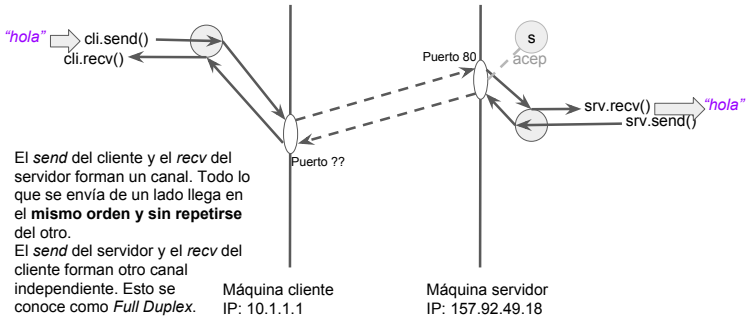
Envio y recepción de datos

Conexión establecida



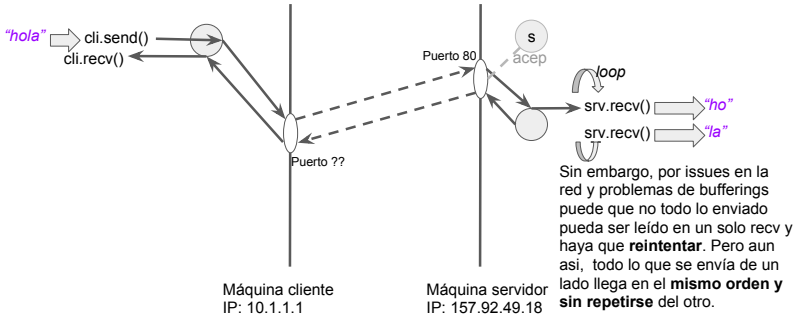
- El socket **`acep`** sigue estando disponible para que el servidor acepte a otras conexiones en paralelo mientras antiende a sus clientes (es independiente del socket **`srv`**)
- Al mismo tiempo, el socket **`srv`** quedo asociado a esa conexión en particular y le permitirá al servidor enviar y recibir mensajes de su cliente.
- Tanto el cliente como el servidor se pueden enviar y recibir mensajes (**`send`**/**`recv`**) entre ellos.
- Los mensajes/bytes enviados con **`cli.send`** son recibidos por el servidor con **`srv.recv`**.
- De igual modo el cliente recibe con **`cli.recv`** los bytes enviados por el servidor con **`srv.send`**.

Envio y recepción de datos

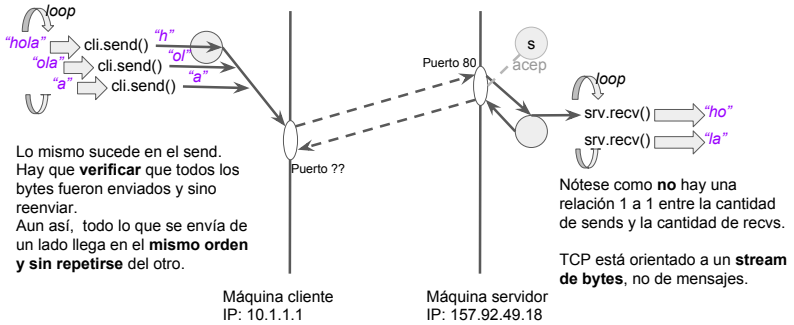


- El par **`cli.send`**–**`srv.recv`** forma un canal en una dirección mientras que el par **`srv.send`**–**`cli-recv`** forma otro canal en el sentido opuesto.
- Ambos canales son independientes. Esto se lo conoce como comunicación Full Duplex
- TCP garantiza que los bytes enviados llegaran en el mismo orden, sin repeticiones y sin pérdidas del otro lado.
- Otro protocolos como UDP no son tan robustos...

Envio y recepción de datos en la realidad



Envio y recepción de datos en la realidad

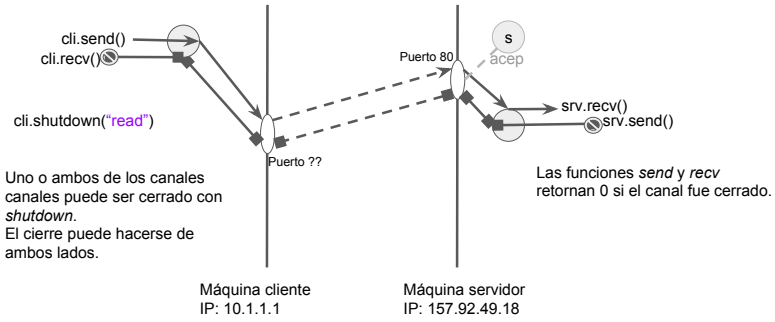


- Sin embargo TCP NO garantiza que todos los bytes pasados a **`send`** se puedan enviar en un solo intento: el programador debera hacer múltiples llamadas a **`send`**.
- De igual modo, no todo lo enviado sera recibido en una única llamada a **`recv`**: el programador debera hacer múltiples llamadas a **`recv`**.

Canal de comunicación TCP

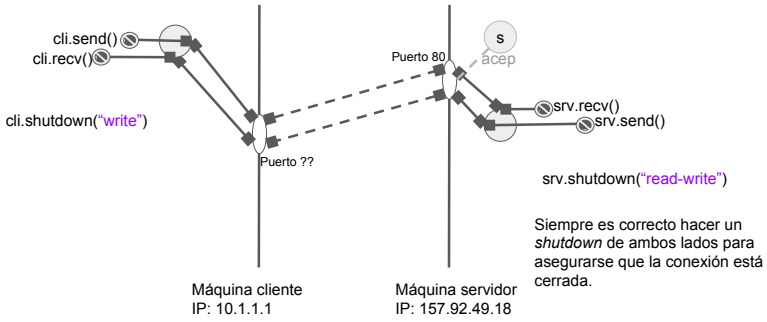
Finalización de un canal

Cierre de conexión parcial

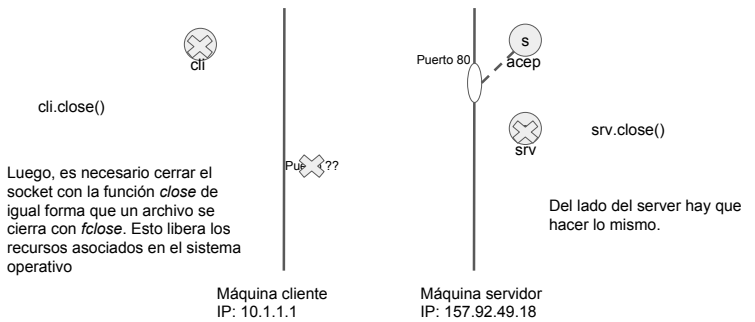


- Parcial en un sentido (envio) **`SHUT_WR`**
- Parcial en el otro sentido (recepción) **`SHUT_RD`**
- Total en ambos sentidos **`SHUT_RDWR`**

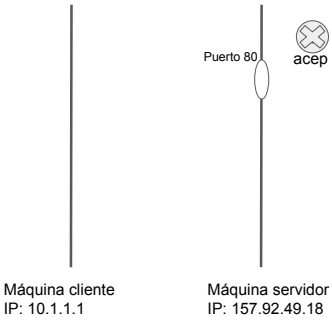
Cierre de conexión total



Liberación de los recursos con close



Cierre y liberación del socket aceptador



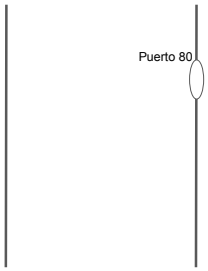
acep.shutdown("read-write")
acep.close()

Finalmente, para cerrar el socket aceptador hay que hacer un *shutdown* y un *close*.

Máquina cliente
IP: 10.1.1.1

Máquina servidor
IP: 157.92.49.18

TIME WAIT



A pesar de haber hecho un *shutdown* y un *close*, el puerto queda en un estado especial llamado TIME_WAIT (incluso si el proceso terminó)

Esto es forzado por el sistema operativo para evitar "ciertos problemas"

Después de unos segundos el puerto queda libre para volver a ser usado.

Máquina cliente
IP: 10.1.1.1

Máquina servidor
IP: 157.92.49.18