

Introducción a Sockets en C

¹ Facultad de Ingeniería
Universidad de Buenos Aires



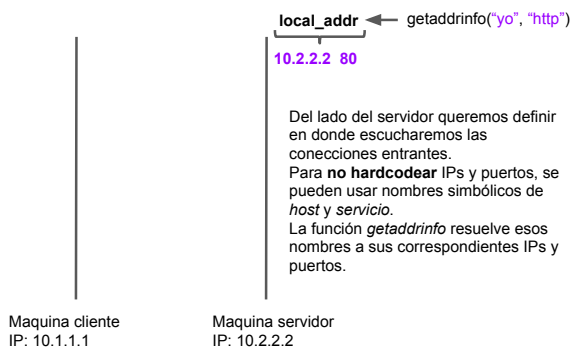
De qué va esto?

- 1 Resolución de nombres
- 2 Sockets
 - Establecimiento de un canal
 - Canal de comunicación
 - Finalización de la comunicación
- 3 Netcat and Netstat



Resolución de nombres

No hardcodeen las IPs ni los puertos



Resolución de nombres

No hardcodeen las IPs ni los puertos

`remote_addr ← getaddrinfo("f.uba", "http")`
`10.2.2.2 80`

Del lado del cliente queremos definir a quien nos queremos conectar.

Maquina cliente
IP: 10.1.1.1

Maquina servidor
IP: 10.2.2.2

Resolución de nombres

No hardcodeen las IPs ni los puertos

```
1 struct addrinfo hints, *results, *addr;
2
3 int status = getaddrinfo(
4     hostname,
5     service,
6     &hints, // <- el filtro
7     &results // <- lista de direcciones
8 );
9
10 addr = results[0]; // hacer algo mas robusto aqui!
```



Resolución de nombres

```
1 |memset(&hints, 0, sizeof(struct addrinfo));
2 |hints.ai_family   = AF_INET;      /* IPv4 */
3 |hints.ai_socktype = SOCK_STREAM;  /* TCP */
4 |hints.ai_flags    = 0;
```

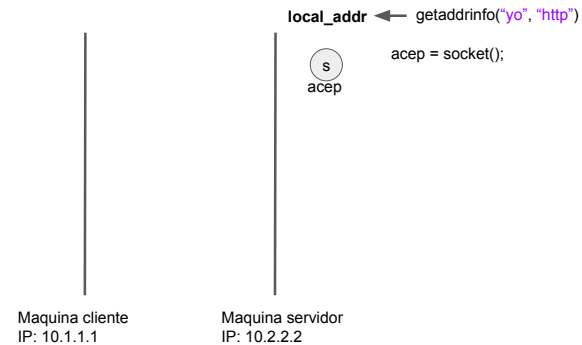
- Familia `AF_UNIX`
 - Para la comunicación entre procesos locales.
- Familias `AF_INET` (IPv4) y `AF_INET6` (IPv6)
 - Para la comunicación a través de la Internet.
- Tipo `SOCK_DGRAM` (UDP).
 - Sin conexión. Orientado a mensajes. Los mensajes se pierden, duplican y llegan en desorden.
- Tipo `SOCK_STREAM` (TCP).
 - Con conexión, full-duplex. Orientado al streaming. Los bytes llegan en orden y sin pérdidas.
- Flags: `0` (socket cliente); `AI_PASSIVE` (socket aceptador)



7542 Introducción a Sockets en C

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

Creación de un socket



7542 Introducción a Sockets en C

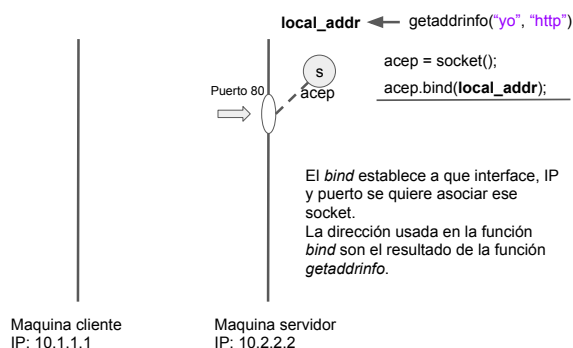
Creación de un socket

```
1 |int skt = socket (
2 |    addr->ai_family,
3 |    addr->ai_socktype,
4 |    addr->ai_protocol
5 |    );
```



7542 Introducción a Sockets en C

Enlazado de un socket a una dirección



7542 Introducción a Sockets en C

- 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.

Enlazado de un socket a una dirección

```
1 int status = bind(  
2     skt,  
3     addr->ai_addr,  
4     addr->ai_addrlen  
5 );
```

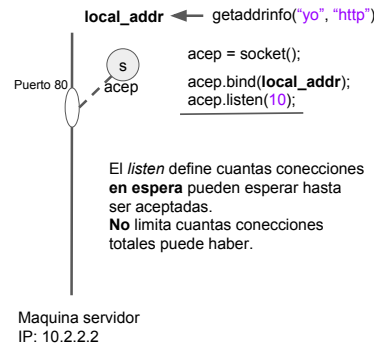


7542

Introducción a Sockets en C

- 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.

Socket aceptador o también conocido como el socket pasivo



7542

Introducción a Sockets en C

Socket aceptador o también conocido como el socket pasivo

```
1 int status = listen(  
2     skt,  
3     10  
4 );  
5  
6 int srv = accept(  
7     skt,  
8     NULL,  
9     NULL  
10 );
```



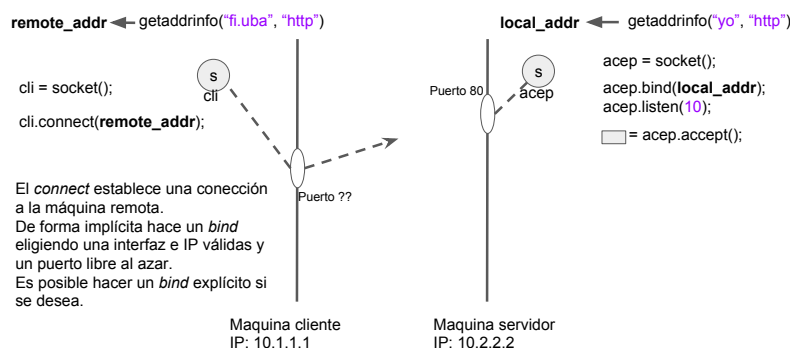
7542

Introducción a Sockets en C

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

Conexión con el servidor

Estableciendo conexión



7542

Introducción a Sockets en C

Conección con el servidor

Estableciendo conexión

```
1 int status = connect (
2     skt,
3     addr->ai_addr,
4     addr->ai_addrlen
5 );
```



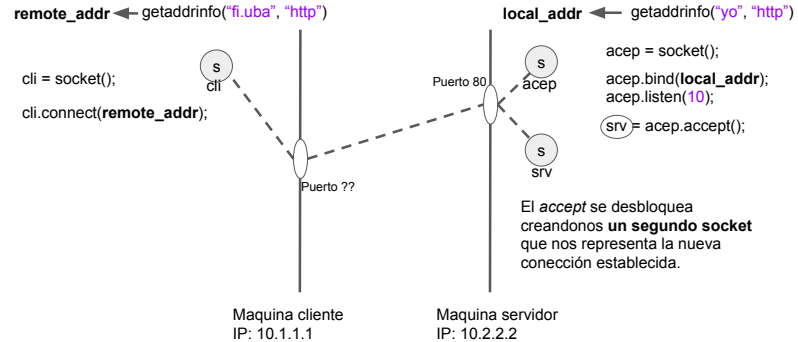
7542

Introducción a Sockets en C

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

Conección con el servidor

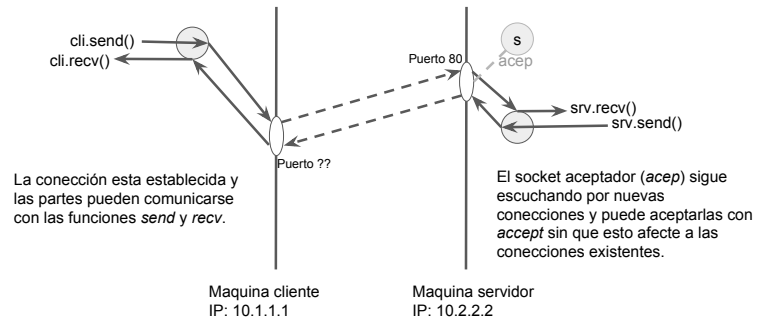
Aceptando la conexión



7542

Introducción a Sockets en C

Conección establecida

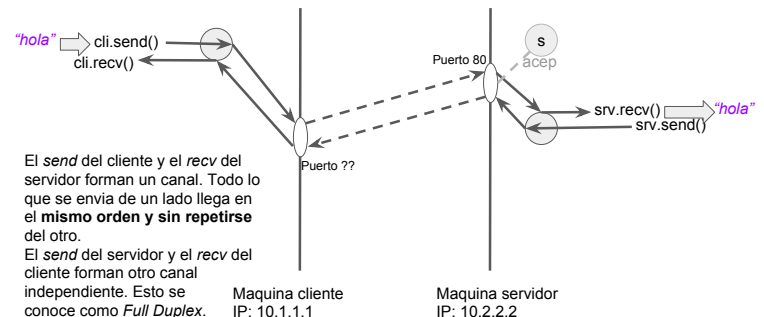


7542

Introducción a Sockets en C

Envío y recepción de datos

- El socket `acep` sigue estando disponible para que el servidor acepte a otras conexiones en paralelo mientras atiende 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`.

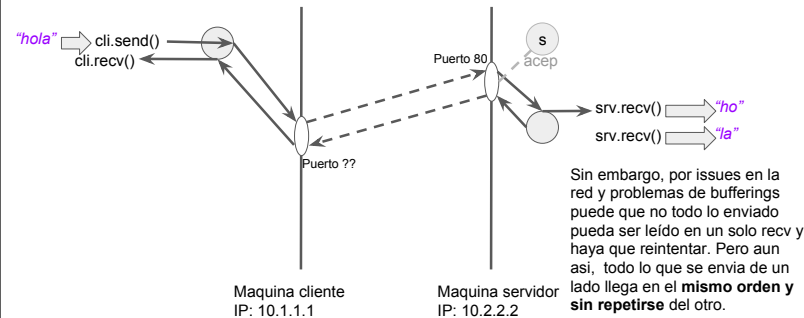


7542

Introducción a Sockets en C

- 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...

Envío y recepción de datos en la realidad



Recepción de datos incremental

```

1 char buf[MSG_LEN]; // buffer donde guardar los datos
2 int bytes_recv = 0;
3
4 while (MSG_LEN > bytes_recv && skt_still_open) {
5     s = recv(skt, &buf[bytes_recv], MSG_LEN - bytes_recv - 1,
6               MSG_NOSIGNAL);
7     if (s < 0) { // Error inesperado
8         /* ... */
9     }
10    else if (s == 0) { // Nos cerraron el socket
11        /* ... */
12    }
13    else {
14        bytes_recv += s;
15    }
16 }

```



Envío y recepción de datos

```

1 int s = send(skt,
2             buf,
3             bytes_to_sent,
4             flags // MSG_NOSIGNAL
5             );
6
7 int s = recv(skt,
8             buf,
9             bytes_to_recv,
10            flags // MSG_NOSIGNAL
11            );
12
13 (s < 0) // Error inesperado
14 (s == 0) // El socket fue cerrado
15 (s > 0) // Ok: s bytes fueron enviados/recibidos

```



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

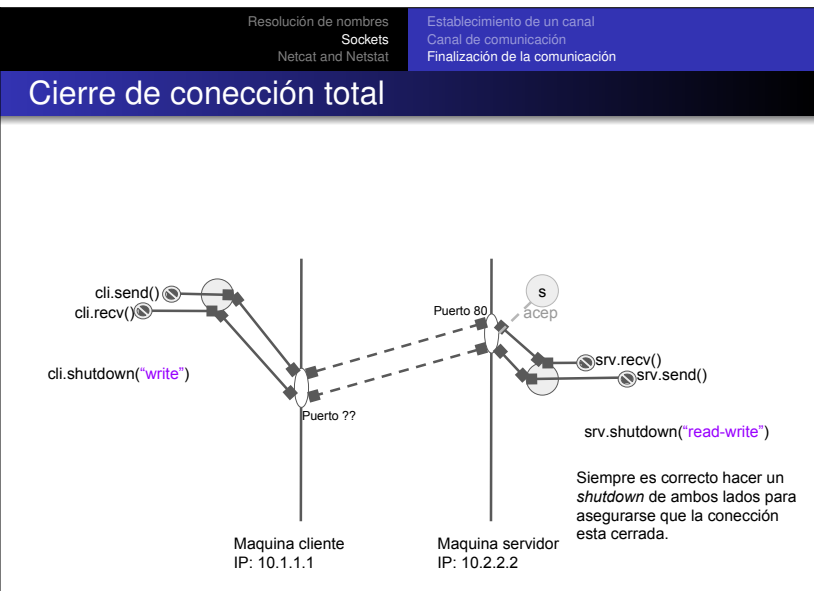
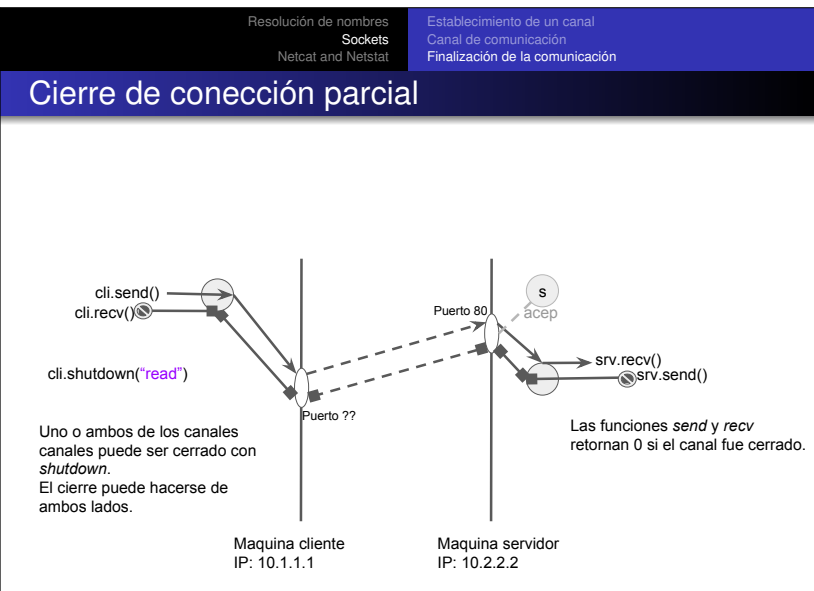
Envío de datos incremental

```

1 char buf[MSG_LEN]; // buffer con los datos a enviar
2 int bytes_sent = 0;
3
4 while (MSG_LEN > bytes_sent && skt_still_open) {
5     s = send(skt, &buf[bytes_sent], MSG_LEN - bytes_sent,
6              MSG_NOSIGNAL);
7     if (s < 0) { // Error inesperado
8         /* ... */
9     }
10    else if (s == 0) { // Nos cerraron el socket
11        /* ... */
12    }
13    else {
14        bytes_sent += s;
15    }
16 }

```



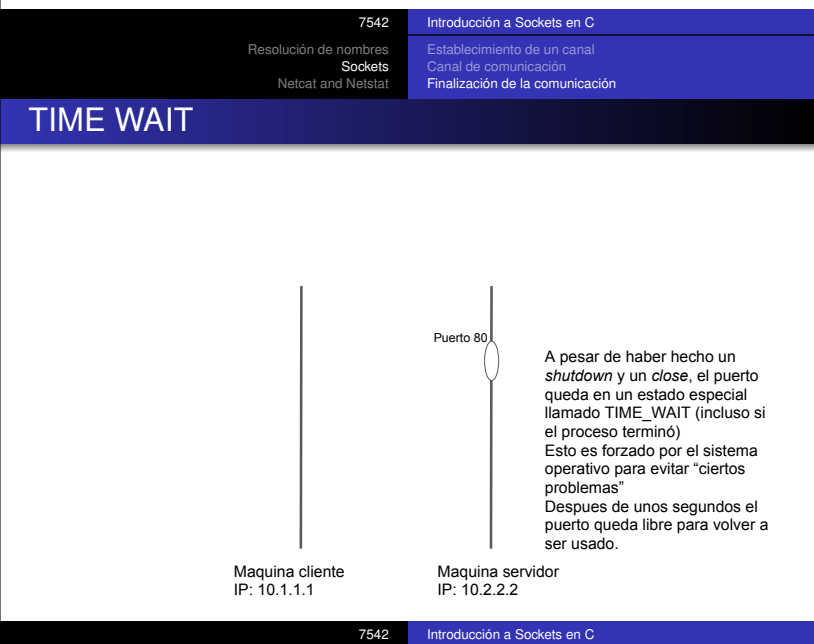
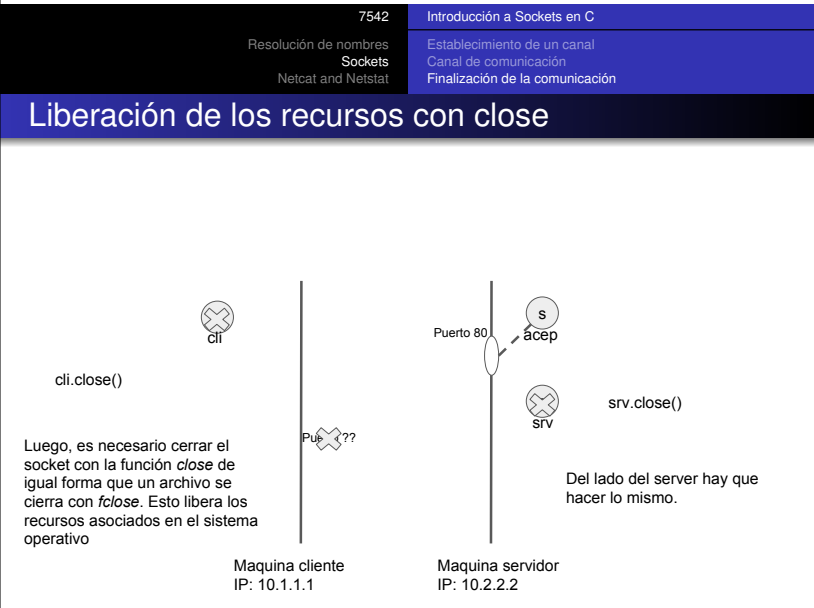



| | |
|---|---|
| 7542 | Introducción a Sockets en C |
| Resolución de nombres Sockets Netcat and Netstat | Establecimiento de un canal Canal de comunicación Finalización de la comunicación |

Cierre de conexión

```
1 int status = shutdown(  
2     skt,  
3     how  
4 );
```

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



| | | | | | |
|--|--|---|--|--|--|
| Resolución de nombres Sockets Netcat and Netstat | | Establecimiento de un canal Canal de comunicación Finalización de la comunicación | | Resolución de nombres Sockets Netcat and Netstat | |
| TIME WAIT -> Reuse Address | | | | Netcat | |
| <p>Si el puerto 80 esta en el estado TIME WAIT, esto termina en error (Address Already in Use):</p> <pre>1 int acep = socket(...); 2 int status = bind(acep, ...); //bind al puerto 80</pre> <p>La solución es configurar al socket aceptador para que pueda reusar la dirección:</p> <pre>1 int acep = socket(...); 2 3 int val = 1; 4 setsockopt(acep, SOL_SOCKET, SO_REUSEADDR, &val, sizeof(val)); 5 6 int status = bind(acep, ...); //bind al puerto 80</pre> | | | | <pre>1 nc -l 8080 # server mode (one time) 2 3 nc -k -l 8080 # server mode (N times) 4 5 nc 127.0.0.1 8080 # client mode</pre> | |
| 7542 | | Introducción a Sockets en C | | 7542 | |
| Resolución de nombres Sockets Netcat and Netstat | | | | Resolución de nombres Sockets Netcat and Netstat | |
| Netcat | | | | Netstat | |
| Una forma simple de copiar archivos entre máquinas | | | | | |
| <pre>1 machineA\$ nc -l 8080 > file_copied 1 machineB\$ nc machineA_ip 8080 < to_copy_file</pre> | | | | <pre>1 machineA\$ nc -l 1234 & 2 machineA\$ nc -l 8080 & 3 machineA\$ nc 127.0.0.1 8080 & 4 5 machineA\$ netstat -tauon 6 Active Internet connections (servers and established) 7 Proto Local Address Foreign Address State 8 tcp 127.0.0.1:1234 0.0.0.0:* LISTEN 9 tcp 127.0.0.1:8080 127.0.0.1:33036 ESTABLISHED 10 tcp 127.0.0.1:33036 127.0.0.1:8080 ESTABLISHED</pre> | |
| 7542 | | Introducción a Sockets en C | | 7542 | |
| Resolución de nombres Sockets Netcat and Netstat | | | | Appendix | |
| Netstat | | | | Referencias I | |
| <pre>1 machineA\$ sudo killall -9 nc 2 3 machineA\$ netstat -tauon 4 Active Internet connections (servers and established) 5 Proto Local Address Foreign Address State 6 tcp 127.0.0.1:8080 127.0.0.1:33036 TIME_WAIT</pre> | | | | <div> man getaddrinfo</div> <div> man netcat</div> <div> man netstat</div> | |
| 7542 | | Introducción a Sockets en C | | 7542 | |
| | | | | Introducción a Sockets en C | |