

# Esercitazione 2020-10-09

## Scambio di messaggi client-server TCP - Esercizio 1.5

In questo esercizio dobbiamo inviare dal client al server un username, ed il server deve rispondere con una stringa del tipo "follen, sei il 2° utente del server!".

In aggiunta, dobbiamo inviare **solo i byte strettamente necessari** al server, quindi è necessario inviare prima un messaggio di informazione che dice al server quanti byte riceverà, e poi inviare i dati effettivi:

### Lato client

```
printf("Inserisci il tuo nome: "); scanf("%s", buf);
int len = strlen(buf);

write(sd, &len, sizeof(len));

printf("Byte trasmessi: %i\n", write(sd, &buf, strlen(buf)));
```

Andiamo quindi a leggere da tastiera l'username e ne calcoliamo l'effettiva lunghezza. Effettuiamo una prima write al server dove inviamo la lunghezza dell'username, successivamente inviamo l'username.

### Lato server

```
int len = 0;
read(sd2, &len, sizeof(len));

int n = 0;
while (n < len)
{
    n += read(sd2, buf + n, len - n);
}
```

Lato server andiamo prima a leggere la lunghezza dell'username, dopodichè leggiamo **al più** len bytes. Questo perchè **non sappiamo a priori** la lunghezza dei pacchetti inviati con il protocollo TCP, e dobbiamo quindi **leggere finchè i bytes non sono stati letti totalmente**.

## Cosa vede wireshark?

### Connessione preliminare

Cattura da Adapter for loopback traffic capture (tcp port 5193)

File Modifica Visualizza Vai Cattura Analizza Statistiche Telefonia Wireless Strumenti Aiuto

Applica un filtro di visualizzazione ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	56066 → 5193 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
2	0.000045	127.0.0.1	127.0.0.1	TCP	56	5193 → 56066 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
3	0.000115	127.0.0.1	127.0.0.1	TCP	44	56066 → 5193 [ACK] Seq=1 Ack=1 Win=2619648 Len=0

< >

> Frame 1: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface \Device\NPF\_{Loopback}, id 0

> Null/Loopback

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

> Transmission Control Protocol, Src Port: 56066, Dst Port: 5193, Seq: 0, Len: 0

```

0000  02 00 00 00 45 00 00 34 d4 be 40 00 00 06 00 00  ....E..4..@..
0010  7f 00 00 01 7f 00 00 01 db 02 14 49 55 e3 3c 3c  ....IU<<
0020  00 00 00 00 80 02 ff ff f5 7e 00 00 02 04 ff d7  ....~.....
0030  01 03 03 08 01 01 04 02  ....

```

Adapter for loopback traffic capture: <live capture in progress> Pacchetti: 3 · visualizzati: 3 (100.0%) Profilo: Default

Come sappiamo, con il protocollo TCP viene eseguita una connessione preliminare tra i due endpoint.

## Invio della lunghezza dell'username al server

Cattura da Adapter for loopback traffic capture (tcp port 5193)

File Modifica Visualizza Vai Cattura Analizza Statistiche Telefonia Wireless Strumenti Aiuto

Applica un filtro di visualizzazione ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	56066 → 5193 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
2	0.000045	127.0.0.1	127.0.0.1	TCP	56	5193 → 56066 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
3	0.000115	127.0.0.1	127.0.0.1	TCP	44	56066 → 5193 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
4	83.597335	127.0.0.1	127.0.0.1	TCP	48	56066 → 5193 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=4
5	83.597357	127.0.0.1	127.0.0.1	TCP	44	5193 → 56066 [ACK] Seq=1 Ack=5 Win=2619648 Len=0
6	83.597372	127.0.0.1	127.0.0.1	TCP	50	56066 → 5193 [PSH, ACK] Seq=5 Ack=1 Win=2619648 Len=6
7	83.597381	127.0.0.1	127.0.0.1	TCP	44	5193 → 56066 [ACK] Seq=1 Ack=11 Win=2619648 Len=0
8	83.597408	127.0.0.1	127.0.0.1	TCP	300	5193 → 56066 [PSH, ACK] Seq=1 Ack=11 Win=2619648 Len=256
9	83.597425	127.0.0.1	127.0.0.1	TCP	44	56066 → 5193 [ACK] Seq=11 Ack=257 Win=2619392 Len=0
10	83.597448	127.0.0.1	127.0.0.1	TCP	44	5193 → 56066 [FIN, ACK] Seq=257 Ack=11 Win=2619648 Len=0
11	83.597457	127.0.0.1	127.0.0.1	TCP	44	56066 → 5193 [ACK] Seq=11 Ack=258 Win=2619392 Len=0
12	83.598020	127.0.0.1	127.0.0.1	TCP	44	56066 → 5193 [FIN, ACK] Seq=11 Ack=258 Win=2619392 Len=0
13	83.598047	127.0.0.1	127.0.0.1	TCP	44	5193 → 56066 [ACK] Seq=258 Ack=12 Win=2619648 Len=0

< >

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

> Transmission Control Protocol, Src Port: 56066, Dst Port: 5193, Seq: 1, Ack: 1, Len: 4

▼ Data (4 bytes)

Data: 06000000

[Length: 4]

```

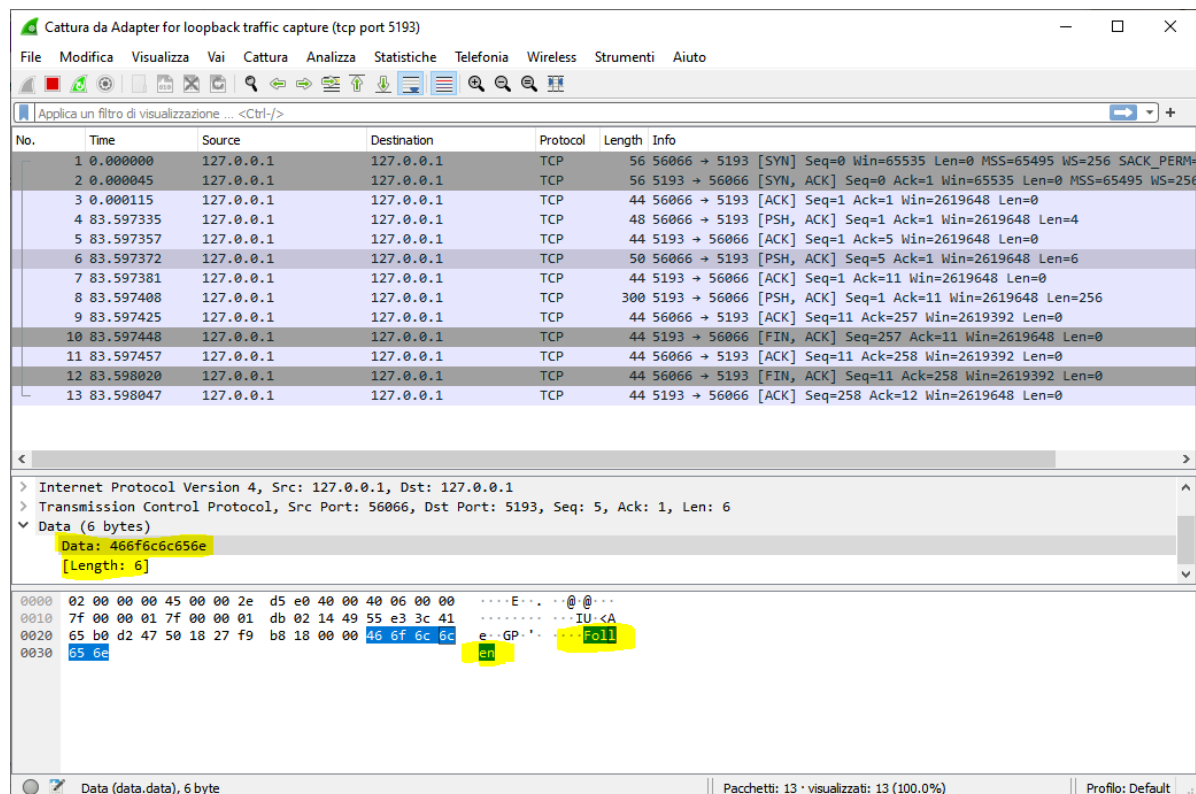
0000  02 00 00 00 45 00 00 2c d5 de 40 00 00 06 00 00  ....E..4..@..
0010  7f 00 00 01 7f 00 00 01 db 02 14 49 55 e3 3c 3d  ....IU<=
0020  65 b0 d2 47 50 18 27 f9 ca 68 00 00 06 00 00 00  e..GP...h.....

```

Adapter for loopback traffic capture: <live capture in progress> Pacchetti: 13 · visualizzati: 13 (100.0%) Profilo: Default

Viene poi inviata la lunghezza della stringa ("follen") al server; infatti i bytes da inviare saranno 6.

## Invio dell'username



Cattura da Adapter for loopback traffic capture (tcp port 5193)

File Modifica Visualizza Vai Cattura Analizza Statistiche Telefonica Wireless Strumenti Aiuto

Applica un filtro di visualizzazione ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	56066 → 5193 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
2	0.000045	127.0.0.1	127.0.0.1	TCP	56	5193 → 56066 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
3	0.000115	127.0.0.1	127.0.0.1	TCP	44	56066 → 5193 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
4	83.597335	127.0.0.1	127.0.0.1	TCP	48	56066 → 5193 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=4
5	83.597357	127.0.0.1	127.0.0.1	TCP	44	5193 → 56066 [ACK] Seq=1 Ack=5 Win=2619648 Len=0
6	83.597372	127.0.0.1	127.0.0.1	TCP	50	56066 → 5193 [PSH, ACK] Seq=5 Ack=1 Win=2619648 Len=6
7	83.597381	127.0.0.1	127.0.0.1	TCP	44	5193 → 56066 [ACK] Seq=1 Ack=11 Win=2619648 Len=0
8	83.597408	127.0.0.1	127.0.0.1	TCP	300	5193 → 56066 [PSH, ACK] Seq=1 Ack=11 Win=2619648 Len=256
9	83.597425	127.0.0.1	127.0.0.1	TCP	44	56066 → 5193 [ACK] Seq=11 Ack=257 Win=2619392 Len=0
10	83.597448	127.0.0.1	127.0.0.1	TCP	44	5193 → 56066 [FIN, ACK] Seq=257 Ack=11 Win=2619648 Len=0
11	83.597457	127.0.0.1	127.0.0.1	TCP	44	56066 → 5193 [ACK] Seq=11 Ack=258 Win=2619392 Len=0
12	83.598020	127.0.0.1	127.0.0.1	TCP	44	56066 → 5193 [FIN, ACK] Seq=11 Ack=258 Win=2619392 Len=0
13	83.598047	127.0.0.1	127.0.0.1	TCP	44	5193 → 56066 [ACK] Seq=258 Ack=12 Win=2619648 Len=0

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

> Transmission Control Protocol, Src Port: 56066, Dst Port: 5193, Seq: 5, Ack: 1, Len: 6

▼ Data (6 bytes)

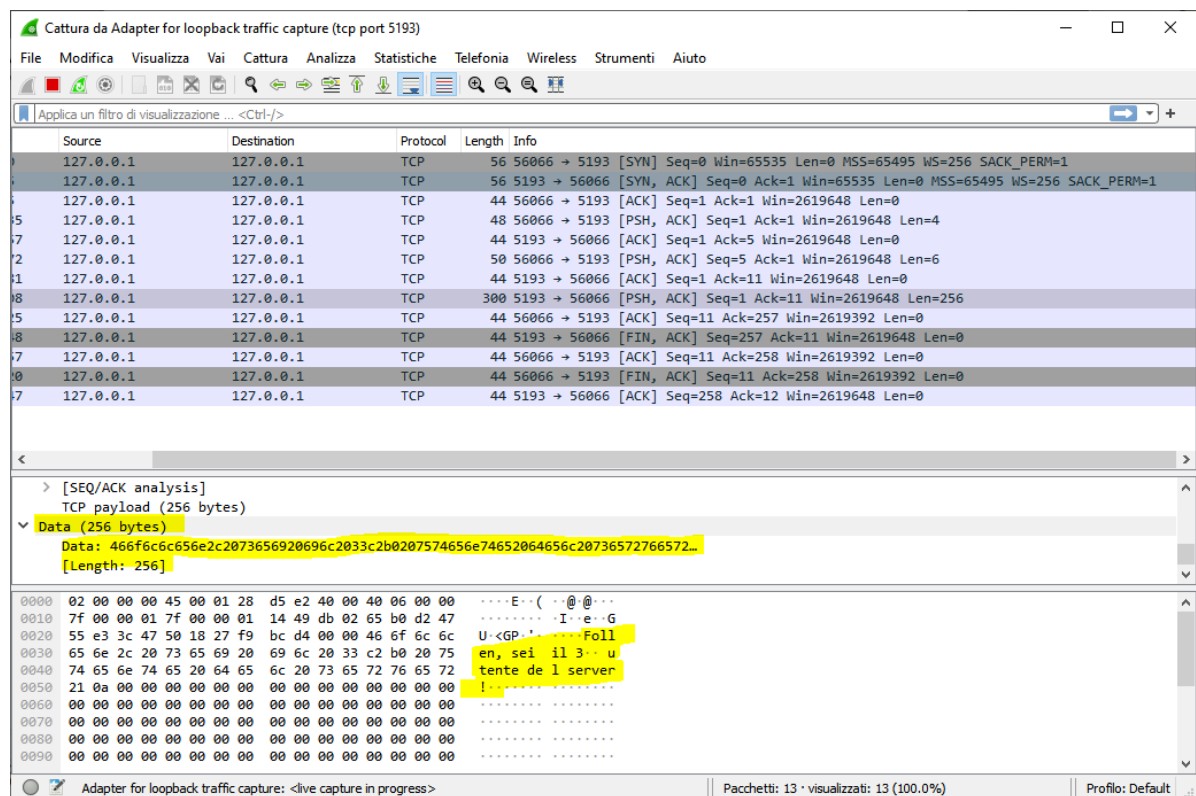
Data: 466f6c6c656e  
[Length: 6]

0000 02 00 00 00 45 00 00 2e d5 e0 40 00 00 06 00 00 ...E... ..@...  
0010 7f 00 00 01 7f 00 00 01 db 02 14 49 55 e3 3c 41 .....IU<A  
0020 65 00 d2 47 50 18 27 f9 b8 18 00 00 46 6f 6c 6c e-GP... ..Fo11  
0030 65 6e

Data (data.data), 6 byte Pacchetti: 13 · visualizzati: 13 (100.0%) Profilo: Default

Viene finalmente inviata la stringa contenente l'username.

## Risposta del server



Cattura da Adapter for loopback traffic capture (tcp port 5193)

File Modifica Visualizza Vai Cattura Analizza Statistiche Telefonica Wireless Strumenti Aiuto

Applica un filtro di visualizzazione ... <Ctrl-/>

Source	Destination	Protocol	Length	Info
127.0.0.1	127.0.0.1	TCP	56	56066 → 5193 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
127.0.0.1	127.0.0.1	TCP	56	5193 → 56066 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
127.0.0.1	127.0.0.1	TCP	44	56066 → 5193 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
127.0.0.1	127.0.0.1	TCP	48	56066 → 5193 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=4
127.0.0.1	127.0.0.1	TCP	44	5193 → 56066 [ACK] Seq=1 Ack=5 Win=2619648 Len=0
127.0.0.1	127.0.0.1	TCP	50	56066 → 5193 [PSH, ACK] Seq=5 Ack=1 Win=2619648 Len=6
127.0.0.1	127.0.0.1	TCP	44	5193 → 56066 [ACK] Seq=1 Ack=11 Win=2619648 Len=0
127.0.0.1	127.0.0.1	TCP	300	5193 → 56066 [PSH, ACK] Seq=1 Ack=11 Win=2619648 Len=256
127.0.0.1	127.0.0.1	TCP	44	56066 → 5193 [ACK] Seq=11 Ack=257 Win=2619392 Len=0
127.0.0.1	127.0.0.1	TCP	44	5193 → 56066 [FIN, ACK] Seq=257 Ack=11 Win=2619648 Len=0
127.0.0.1	127.0.0.1	TCP	44	56066 → 5193 [ACK] Seq=11 Ack=258 Win=2619392 Len=0
127.0.0.1	127.0.0.1	TCP	44	56066 → 5193 [FIN, ACK] Seq=11 Ack=258 Win=2619392 Len=0
127.0.0.1	127.0.0.1	TCP	44	5193 → 56066 [ACK] Seq=258 Ack=12 Win=2619648 Len=0

> [SEQ/ACK analysis]  
TCP payload (256 bytes)

▼ Data (256 bytes)

Data: 466f6c6c656e2c2073656920696c2033c2b0207574656e74652064656c20736572766572...  
[Length: 256]

0000 02 00 00 00 45 00 01 28 d5 e2 40 00 00 06 00 00 ...E... ( ..@...  
0010 7f 00 00 01 7f 00 00 01 14 49 db 02 65 b0 d2 47 .....I...e...G  
0020 55 e3 3c 47 50 18 27 f9 bc d4 00 00 46 6f 6c 6c U-<GP... ..Fo11  
0030 65 6e 2c 20 73 65 69 20 69 6c 20 33 c2 b0 20 75 en, sei il 3... u  
0040 74 65 6e 74 65 20 64 65 6c 20 73 65 72 76 65 72 tente de l server  
0050 21 0a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 !...  
0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

Adapter for loopback traffic capture: <live capture in progress> Pacchetti: 13 · visualizzati: 13 (100.0%) Profilo: Default

Il server risponde con una stringa (questa volta di 256 bytes) contenente la risposta.

1:07

## Esercizi 2.x

## Esercizio 2.1

---

Il primo esercizio è una variante degli esercizi visti fin ora, e prevede che il client legga da stdin delle stringhe in **modo continuo**, e che le invii al server, che non fa altro che stampare la stringa ricevuta sullo stdout.

La lettura delle stringhe da parte del client e la spedizione verso il server continua finchè non si leggerà una stringa che finisce con ".".

## Esercizio 2.2 - Peer to Peer

---

L'API che abbiamo usato fin'ora è stata progettata per operare in architettura client-server; questo perchè da lato server invochiamo sulla socket costruita prima la `listen()` e poi l'`accept()`, che non usiamo lato client.

Partiamo da un modello client-server e proviamo a realizzare un canale di comunicazione p2p.

**Il problema:** se facciamo in modo che uno dei processi si ponga in ascolto per ricevere dei messaggi di richiesta dall'altro processo, non potrà inviare messaggi; questo perchè la `listen` è bloccante.

**La soluzione:** invece di usare il canale di comunicazione in maniera bidirezionale (cosa possibile tramite al protocollo TCP), possiamo pensare di sfruttare parzialmente il canale di comunicazione: una volta da A->B, ed un'altra da B->A; comunichiamo quindi in modo alterno.

## Peer To Peer - Chat Half-Duplex

---

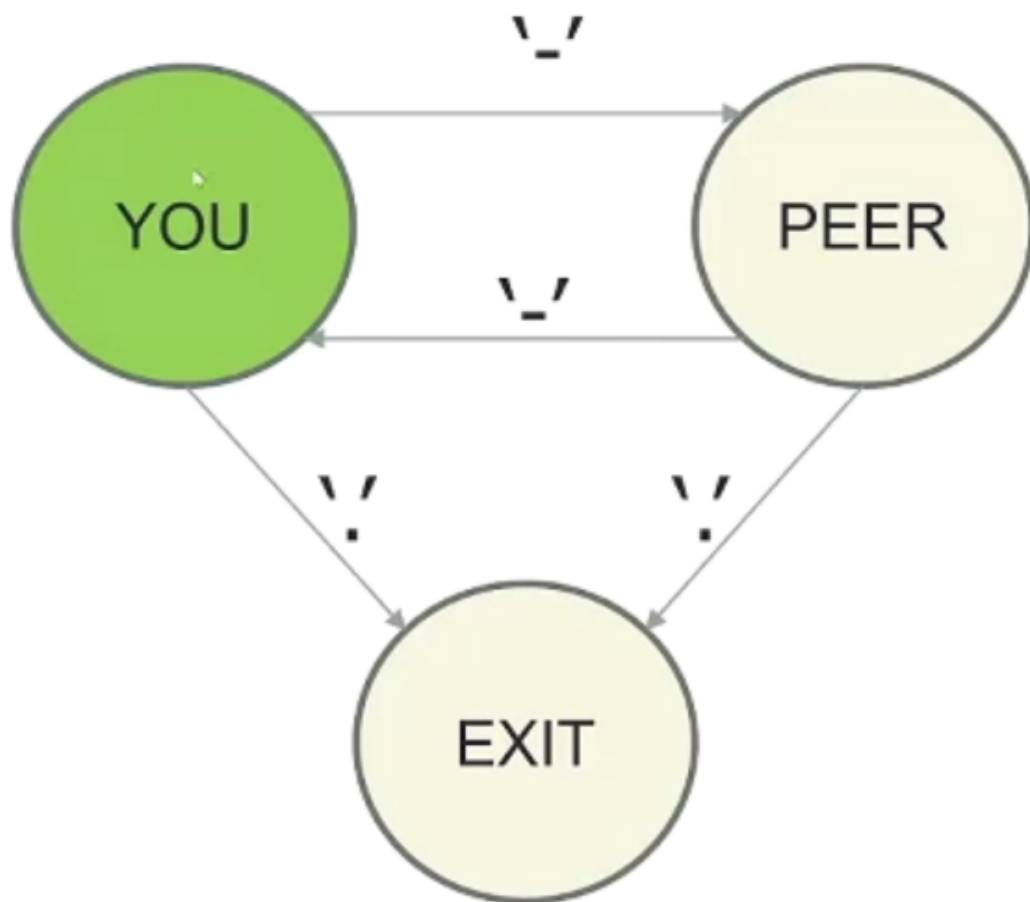
Invece di parlare di server e di client, parliamo di **peer**. Avremo, a seguito della costruzione iniziale del canale (sempre necessaria), il canale di comunicazione TCP diventa un canale che può essere usato in maniera **simmetrica** dai due interlocutori.

Il codice da scrivere per dare vita ai due processi, è praticamente lo stesso; sarà però eseguito in maniera diversa, in modo da tenere in considerazione **lo stato diverso** che i due peer avranno in ogni momento.

# PEER1

```
read <- stdin  
write -> net channel
```

```
read <- net channel  
write -> stdout
```



Quando il peer1 è abilitato a scrivere sul canale, il peer2 è abilitato a leggere. Se i due peer operano in questo stato, non ci sono problemi di "conflitto" sul canale, visto che solo un peer alla volta lo usa per scrivere.

Periodicamente, lo stato cambia; questo vuol dire che quando viene inviato un carattere speciale (ad esempio un trattino), viene cambiato lo stato, quindi se inizialmente il primo peer scriveva ed il secondo leggeva, con l'invio del carattere speciale il primo peer legge ed il secondo scrive. Quando viene inviato un secondo carattere speciale (ad esempio un punto), **da uno dei due peer**, la connessione viene chiusa.

## Il codice

---

La parte più importante del codice di uno dei due processi è sostanzialmente un loop, che caratterizza un automa a stati. Abbiamo tre stati: **stato YOU, stato PEER e stato EXIT**; inizialmente il client è nello stato YOU, mentre il "server" è nello stato PEER.

All'interno del loop principale vi è uno **switch case**, che a seconda dello stato pone il processo nella fase di lettura o scrittura.

## Fase di scrittura - YOU

Durante la fase di scrittura viene letta da tastiera una stringa, ne viene calcolata l'effettiva lunghezza che viene comunicata al "server", e poi viene inviata. Viene controllato l'ultimo carattere della stringa:

- Se l'ultimo carattere è '.' lo status viene cambiato in EXIT.
- Se l'ultimo carattere è '-' lo status viene cambiato in PEER, ed il controllo viene trasferito all'altro processo.
- In tutti gli altri casi non succede nulla.

Viene poi scritta sul canale di comunicazione prima la lunghezza della stringa letta, e poi viene inviata la stringa effettiva, che il server leggerà.

Questo processo viene ripetuto finchè il controllo non viene affidato all'altro processo.

## Fase di lettura - PEER

Anche in questo caso abbiamo un loop; viene letta prima la lunghezza della stringa da ricevere, e poi la stringa stessa. ancora una volta si controlla la stringa:

- Se l'ultimo carattere è '.' lo status viene cambiato in EXIT.
- Se l'ultimo carattere è '-' lo status viene cambiato in YOU, ed il controllo è del processo corrente.
- In tutti gli altri casi non succede nulla.

## Differenza tra i due processi

Ovviamente il codice tra i due processi è molto simile, ma diverso nella fase iniziale; questo perchè inizialmente uno dei due processi deve predisporre a **ricevere una connessione**, quindi se dal client abbiamo una connect, dal "server" avremo la **listen() e poi accept()**.

Dopo questa fase iniziale, il codice è lo stesso.