



POLITECNICO
MILANO 1863



SLIDES
DURANTE IL
LABORATORIO

Fondamenti di Comunicazioni e Internet

Antonio Capone, Matteo Cesana,
Guido Maier, Francesco Musumeci



POLITECNICO
MILANO 1863



Programmazione Socket

Antonio Capone, Matteo Cesana,
Guido Maier, Francesco Musumeci

Attività di laboratorio: Versioni software

Gli esempi mostrati a lezione usano:

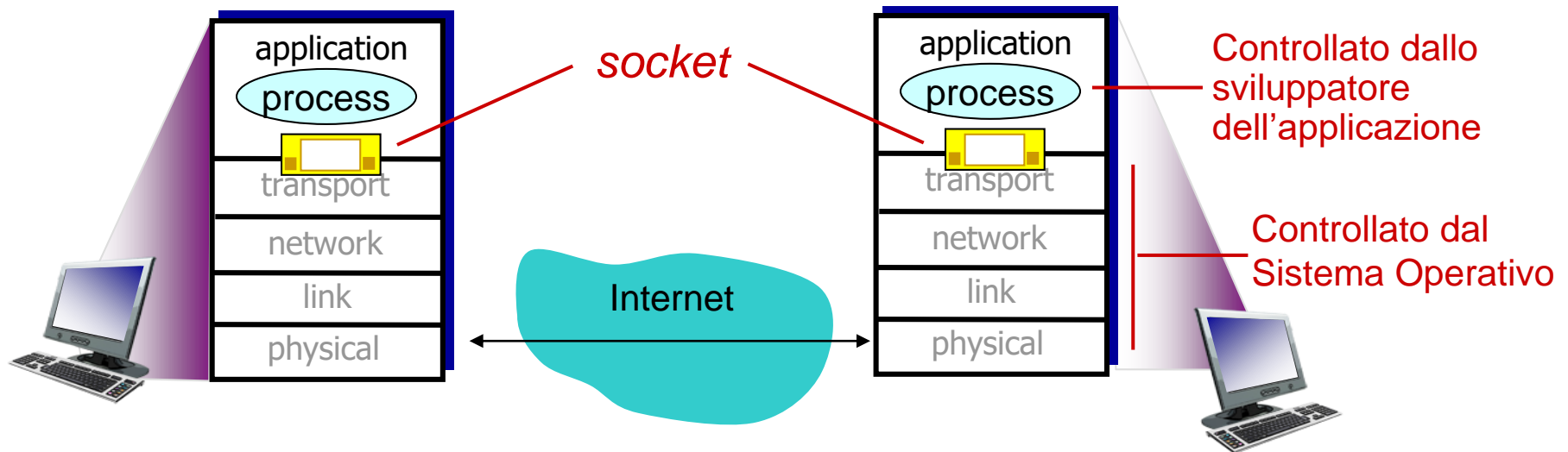
- Python versione 3
- Pycharm IDE education edition
 - include Python 3

Tutti gli esempi sono testati nella macchina locale

- usiamo i socket per far comunicare processi in esecuzione sulla stessa macchina
- gli esempi funzionano altrettanto bene se i processi sono in esecuzione in due macchine distinte

Programmazione Socket

- **Obiettivo:** imparare a sviluppare applicazioni client/server che comunicano utilizzando i sockets
- **Socket:** porta tra il processo applicativo e il protocollo di trasporto end-to-end



Programmazione Socket

API = Application Programming Interface

Socket API

Introdotta in BSD4.1 UNIX, 1981

Creata, utilizzata e rilasciata
esplicitamente dalle applicazioni

Paradigma client/server

Socket API offre due tipi di
servizio di trasporto:

- UDP
- TCP

BSD = Berkeley Software Distribution

socket

È un'interfaccia (porta) *creata
dall'applicazione e controllata
dal SO* attraverso la quale un
processo applicativo può
inviare e ricevere messaggi
a/da un altro processo
applicativo

Programmazione Socket - Basi

- Il server deve essere in esecuzione prima che il client possa inviare dati ad esso (daemon)
- Il server deve avere un socket (porta) attraverso il quale riceve ed invia segmenti
- Allo stesso modo anche il client necessita di un socket
- Il client deve conoscere l'indirizzo IP del server e il numero di porta del processo server

Programmazione Socket *con UDP*

UDP: non c'è “connessione” tra client and server

- Non c'è handshaking
- Il mittente inserisce esplicitamente indirizzo IP e porta destinazione ad ogni segment
- Il SO inserisce l'indirizzo IP e la porta del socket origine ad ogni segmento
- Il server può ricavare indirizzo IP e porta del mittente dai segmenti ricevuti

Punto di vista dell'applicazione

UDP fornisce trasporto non affidabile di gruppi di bytes all'interno di datagrammi scambiati tra client e server

Nota: il termine corretto per “pacchetto UDP” sarebbe “datagramma”, ma in questa lezione useremo indistintamente i termini “segment”, “pacchetto” e “datagramma” UDP.

Esempio

Client:

- L'utente inserisce una riga di testo
- L'applicazione client invia la riga al server

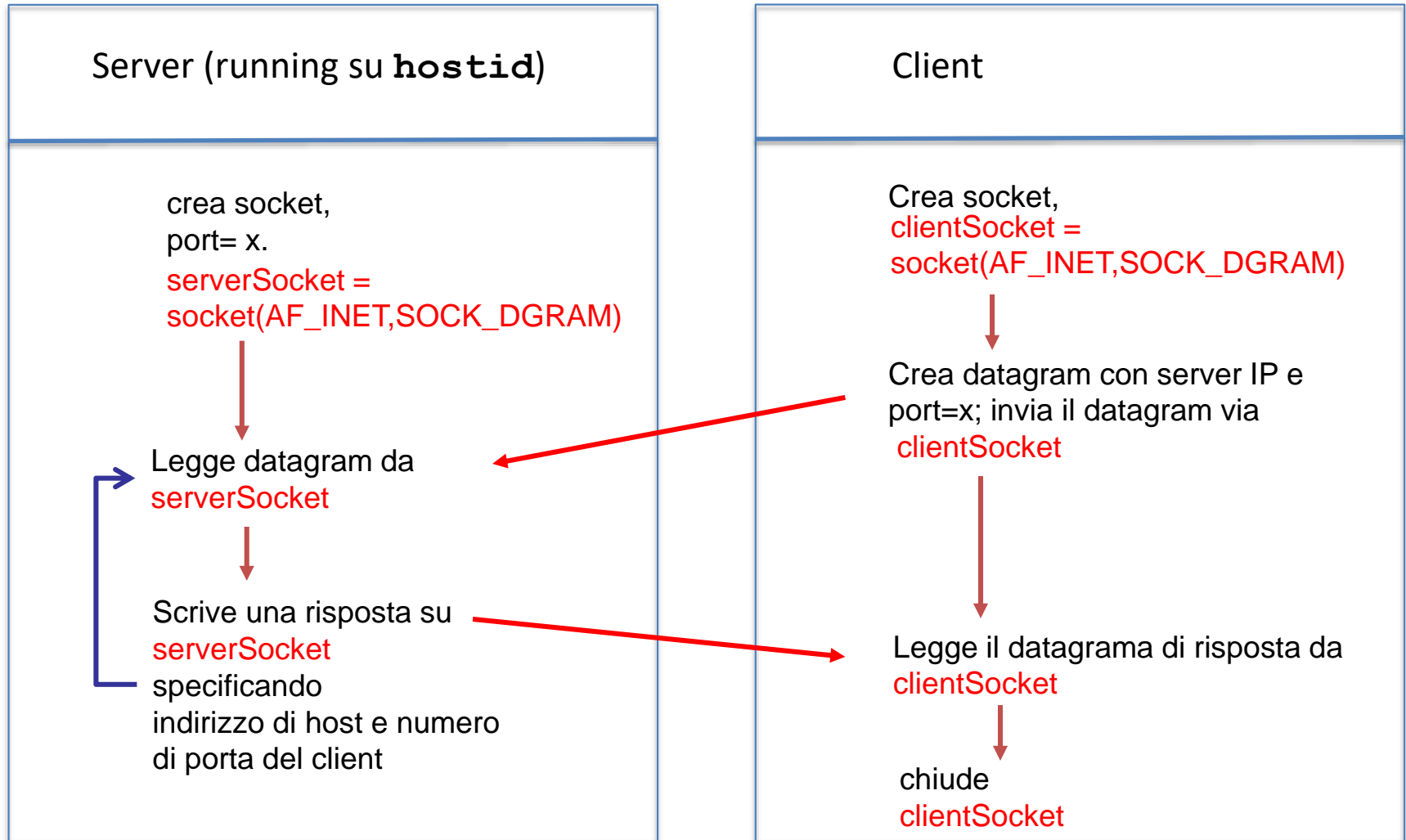
Server:

- Il server riceve la riga di testo
- Rende maiuscole tutte le lettere
- Invia la riga modificata al client

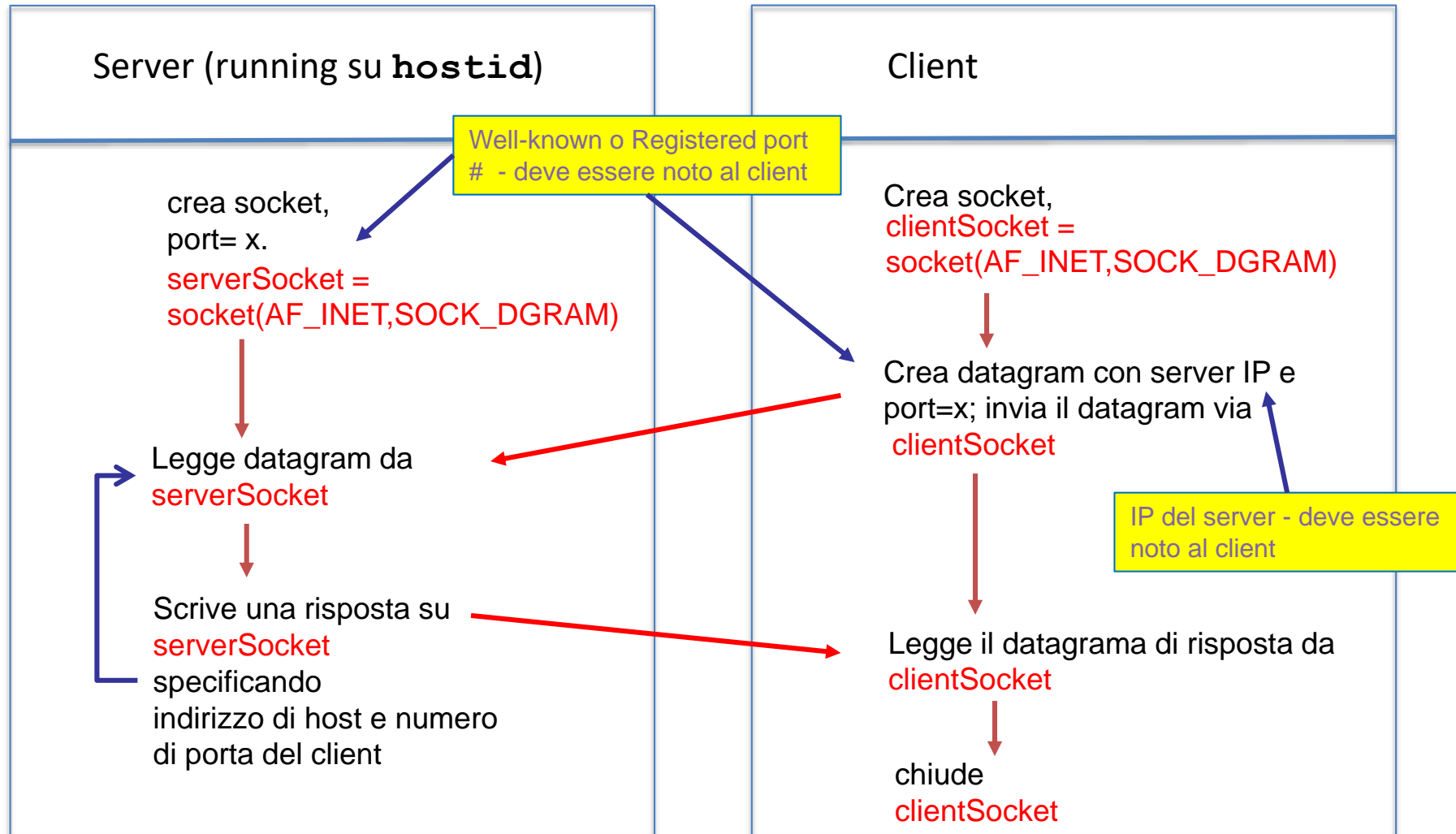
Client:

- Riceve la riga di testo
- La visualizza

Interazione tra socket Client/server: UDP



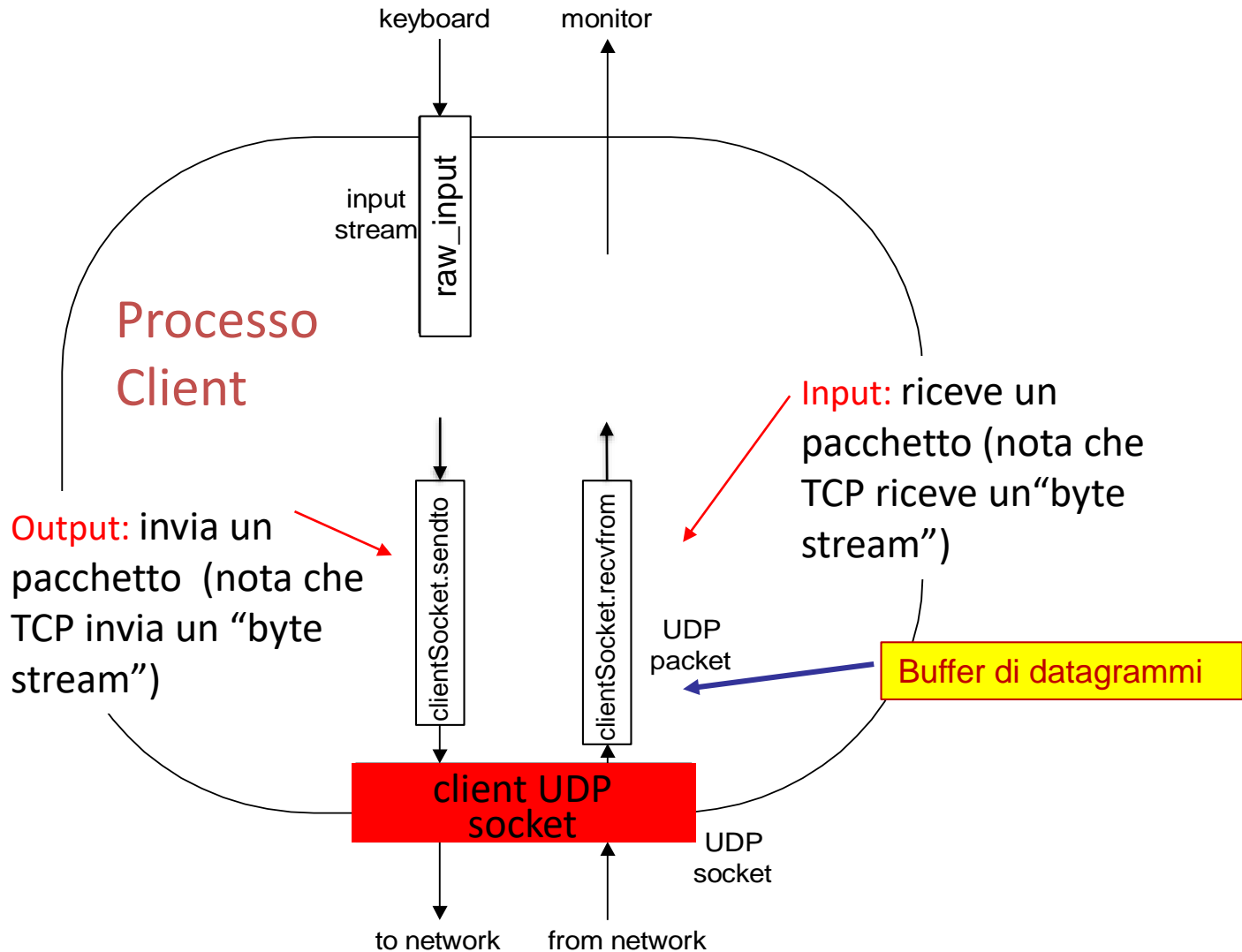
Interazione tra socket Client/server: UDP



`serverSocket = socket(AF_INET, SOCK_DGRAM)`

Nome oggetto (metodo) Nome classe

Esempio: Python client (UDP)



Applicazione esempio: *UDP client*

Include la libreria
Socket di Python

→ `from socket import *`

`serverName = 'localhost'`

Nome simbolico del server

`serverPort = 12000`

12000: # porta processo server

Crea un socket UDP
per il server

→ `clientSocket = socket(AF_INET,`

`SOCK_DGRAM)`

SOCK_DGRAM → UDP

Legge l'input da tastiera

→ `message = input(Inserisci lettere:')`

Aggiunge il nome del server
e la porta al messaggio; lo
invia nel socket

→ `clientSocket.sendto(message.encode('utf-8'),(serverName, serverPort))`

Il socket si pone in ascolto
(l'esecuzione procede quando arriva
un segmento UDP)

Legge i caratteri della
risposta del server dal socket
e li memorizza in una stringa

→ `modifiedMessage, serverAddress =`

`clientSocket.recvfrom(2048)`

Dimensione
del buffer

`modifiedMessage = modifiedMessage.decode('utf-8')`

Mostra la stringa ricevuta
e chiude il socket

→ `Print(modifiedMessage)`

`clientSocket.close()`

La chiamata al DNS per traduzione serverName (hostname) → IP server è fatta dal sistema operativo

Applicazione esempio: *UDP server*

Crea un socket UDP → `from socket import *`

Assegna il socket all porta
locale 12000 →

```
serverPort = 12000  
serverSocket = socket(AF_INET, SOCK_DGRAM)  
serverSocket.bind(('', serverPort))  
Print("Il server è pronto a ricevere")
```

12000: registered port #

loop infinito →

```
while 1:
```

Il socket si pone in ascolto
(l'esecuzione procede quando
arriva un segmento UDP)

```
message, clientAddress =
```

Legge dal socket UDP
message, memorizzando
l'indirizzo del client (IP e porta) →

```
serverSocket.recvfrom(2048)
```

Dimensione
del buffer

```
print('Datagramma da: ', clientAddress)
```

```
message = message.decode('utf-8')
```

```
modifiedMessage = message.upper()
```

I dati per
costruire la
risposta sono
esplicitamente
ricavati dal
pacchetto UDP
ricevuto

Risponde al client con la
stringa tutta in maiuscolo →

```
serverSocket.sendto
```

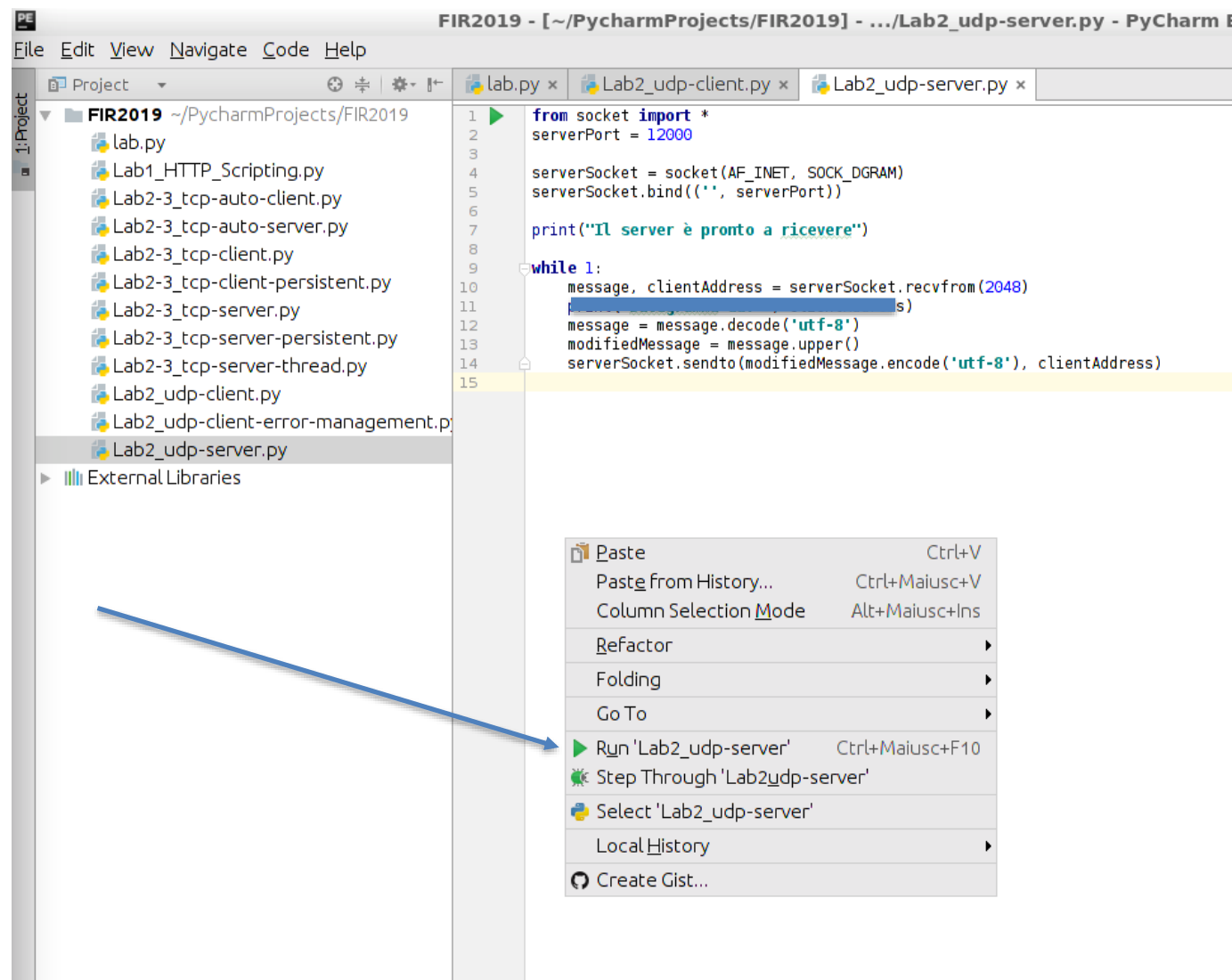
Elaborazione strettamente seriale
dei pacchetti

```
(modifiedMessage.encode('utf-8'), clientAddress)
```

UDP: osservazioni e domande

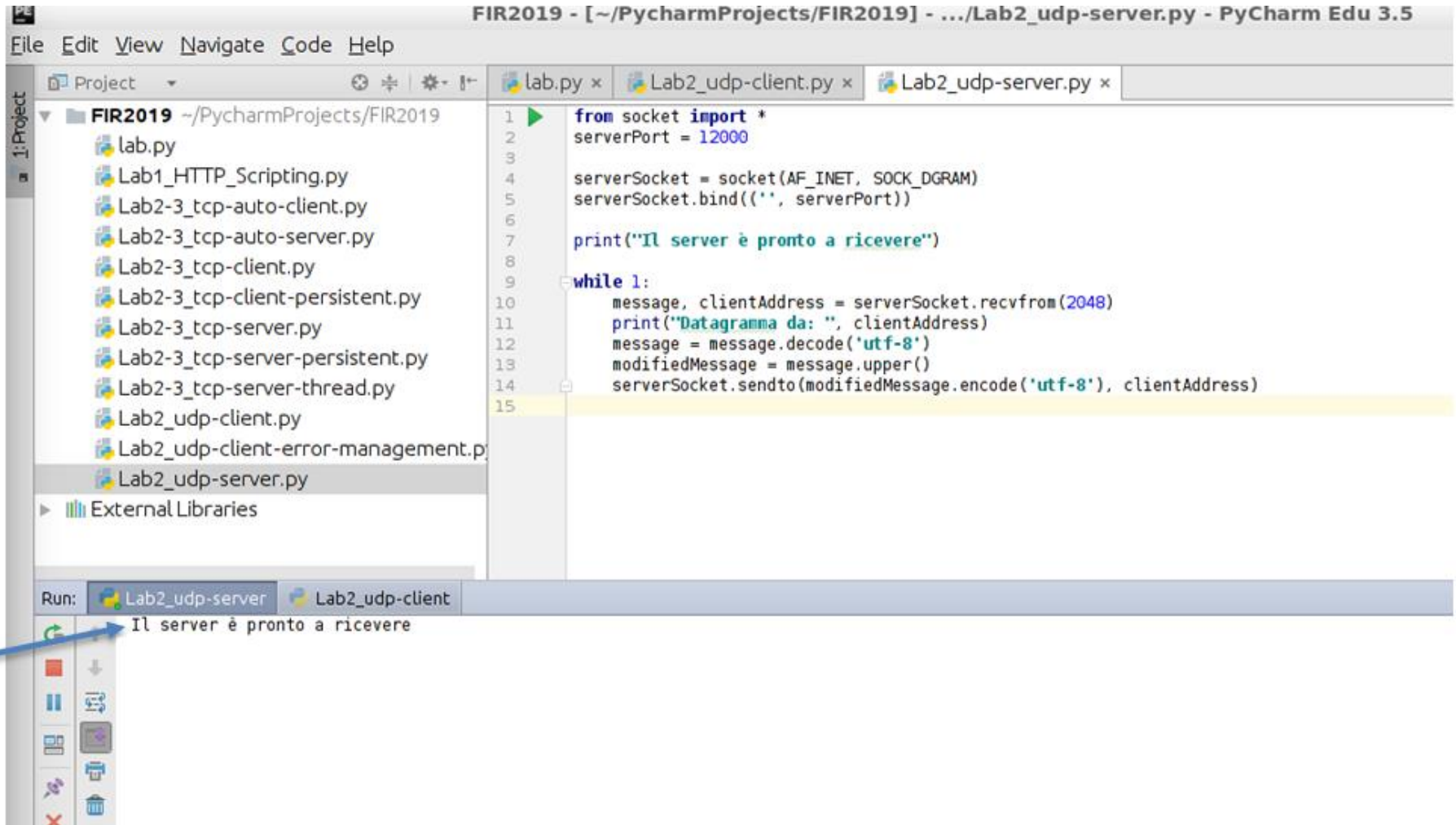
- Client e server utilizzando entrambi DatagramSocket
- IP destinazione e porta sono esplicitamente inseriti nel segmento
- Il client non può inviare un segmento al server senza conoscere l'indirizzo IP e la porta del server
- Il server può essere utilizzato da più di un client, ma tutti invieranno i pacchetti tramite lo stesso socket
 - La separazione dei dati da/a diversi client è un compito che spetta interamente al server

Eseguire il server



In pycharm il server si può lanciare dal menu contestuale (tasto destro sul nome dello script) oppure con la sequenza di tasti **Ctrl+Maiusc+F10**

Messaggi del server



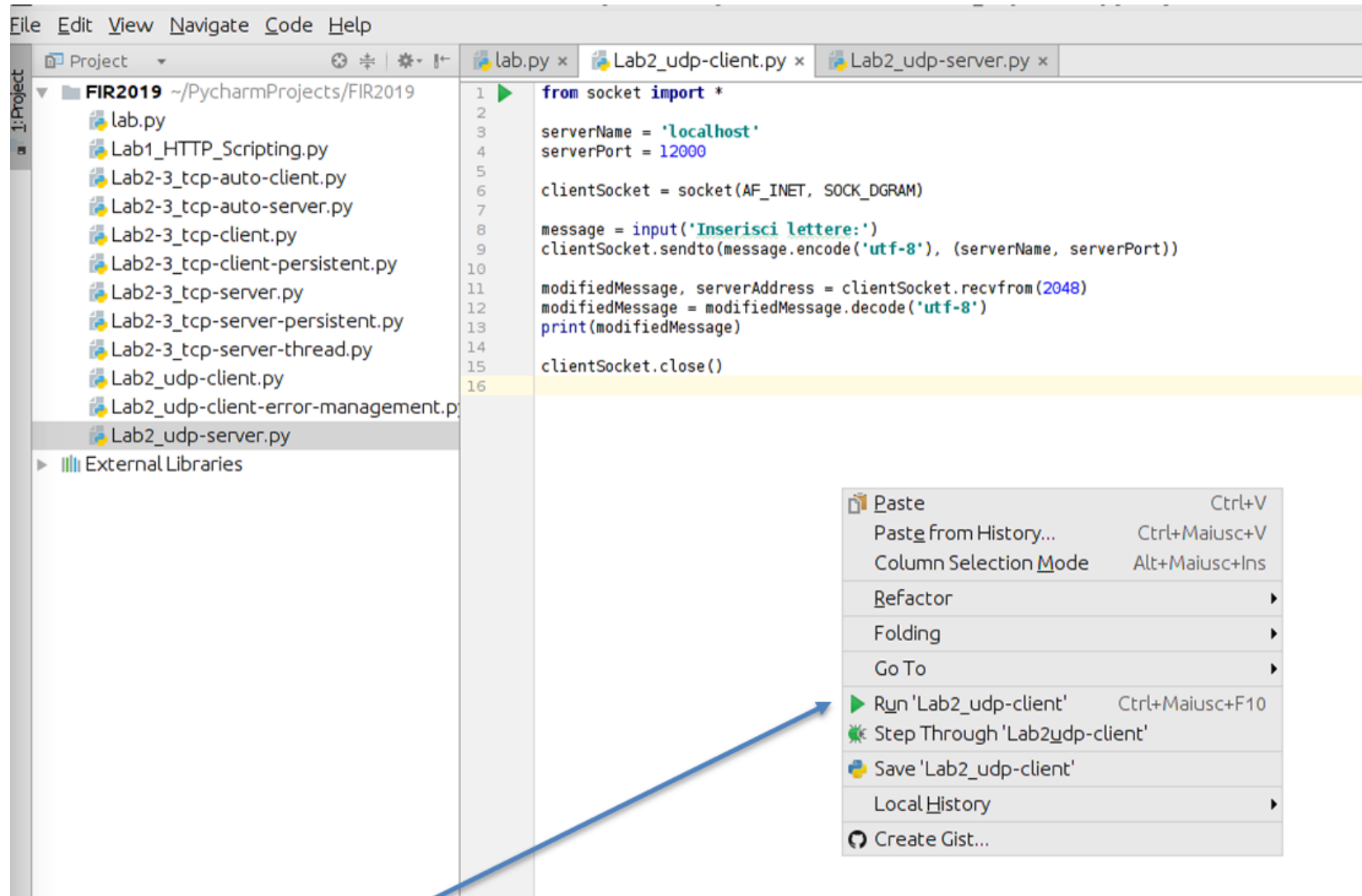
Verifica nell'elenco dei processi: netstat

```
user@user-VirtualBox: ~  
File Modifica Schede Aiuto  
user@user-VirtualBox:~$ netstat -aut  
Connessioni Internet attive (server e stabiliti)  
Proto CodaRic CodaInv Indirizzo locale      Indirizzo remoto      Stato  
tcp      0      0 localhost:63342      *:*                   LISTEN  
tcp      0      0 *:pop3              *:*                   LISTEN  
tcp      0      0 user-VirtualBox:domain *:*                   LISTEN  
tcp      0      0 localhost:ipp        *:*                   LISTEN  
tcp      0      0 *:smtp              *:*                   LISTEN  
tcp      0      0 localhost:6942       *:*                   LISTEN  
tcp      86      0 10.0.2.15:42476      mail.stepik.org:https CLOSE_WAIT  
tcp6     0      0 [::]:http           [::]:*                LISTEN  
tcp6     0      0 [::]:ftp             [::]:*                LISTEN  
tcp6     0      0 ip6-localhost:ipp   [::]:*                LISTEN  
tcp6     0      0 [::]:smtp            [::]:*                LISTEN  
udp      0      0 *:12000             *:*                   LISTEN  
udp      0      0 *:27367             *:*                   LISTEN  
udp      0      0 user-VirtualBox:domain *:*                   LISTEN  
udp      0      0 *:bootpc            *:*                   LISTEN  
udp      0      0 10.0.2.15:ntp        *:*                   LISTEN  
udp      0      0 localhost:ntp        *:*                   LISTEN  
udp      0      0 *:ntp               *:*                   LISTEN  
udp6     0      0 [::]:44966          [::]:*                LISTEN  
udp6     0      0 fe80::a00:27ff:feae:ntp [::]:*                LISTEN  
udp6     0      0 ip6-localhost:ntp   [::]:*                LISTEN  
udp6     0      0 [::]:ntp             [::]:*                LISTEN  
user@user-VirtualBox:~$
```

Comando:
netstat -aut

Il server è in
esecuzione

Eseguire il client



Analogamente al server anche il client si può lanciare dal menu contestuale (tasto destro sul nome dello script).

Inserimento input

The screenshot displays the PyCharm IDE interface. The left sidebar shows a project named 'FIR2019' with a file tree containing various Python scripts, including 'Lab2_udp-client.py'. The main editor window shows the code for 'Lab2_udp-client.py', which is a UDP client program. The code includes imports, server address and port definitions, socket creation, message input, sending, receiving, and printing. The bottom status bar shows the 'Run' configuration for 'Lab2_udp-client', and the console output displays the prompt 'Inserisci lettere:'.

```
1 from socket import *
2
3 serverName = 'localhost'
4 serverPort = 12000
5
6 clientSocket = socket(AF_INET, SOCK_DGRAM)
7
8 message = input('Inserisci lettere:')
9 clientSocket.sendto(message.encode('utf-8'), (serverName, serverPort))
10
11 modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
12 modifiedMessage = modifiedMessage.decode('utf-8')
13 print(modifiedMessage)
14
15 clientSocket.close()
16
```

Inserire la frase da convertire

Run: Lab2_udp-server Lab2_udp-client

Inserisci lettere:

Ricezione e visualizzazione output

The image shows a PyCharm IDE window with a project named 'FIR2019'. The file explorer on the left lists several Python files, including 'Lab2_udp-client.py' and 'Lab2_udp-server.py'. The main editor displays the code for 'Lab2_udp-client.py', which is a UDP client program. The code imports the 'socket' module, sets the server name to 'localhost' and the server port to 12000, creates a client socket, sends a message, receives a response, and prints it. The output window at the bottom shows the execution of the 'Lab2_udp-client' process, displaying the input 'ciao' and the received output 'CIAO'. A blue arrow points from the word 'Output' to the output window.

```
File Edit View Navigate Code Help
Project ~:/PycharmProjects/FIR2019
lab.py Lab1_HTTP_Scripting.py Lab2-3_tcp-auto-client.py Lab2-3_tcp-auto-server.py Lab2-3_tcp-client.py Lab2-3_tcp-client-persistent.py Lab2-3_tcp-server.py Lab2-3_tcp-server-persistent.py Lab2-3_tcp-server-thread.py Lab2_udp-client.py Lab2_udp-client-error-management.py Lab2_udp-server.py
External Libraries

1 from socket import *
2
3 serverName = 'localhost'
4 serverPort = 12000
5
6 clientSocket = socket(AF_INET, SOCK_DGRAM)
7
8 message = input('Inserisci lettere:')
9 clientSocket.sendto(message.encode('utf-8'), (serverName, serverPort))
10
11 modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
12 modifiedMessage = modifiedMessage.decode('utf-8')
13 print(modifiedMessage)
14
15 clientSocket.close()
16
```

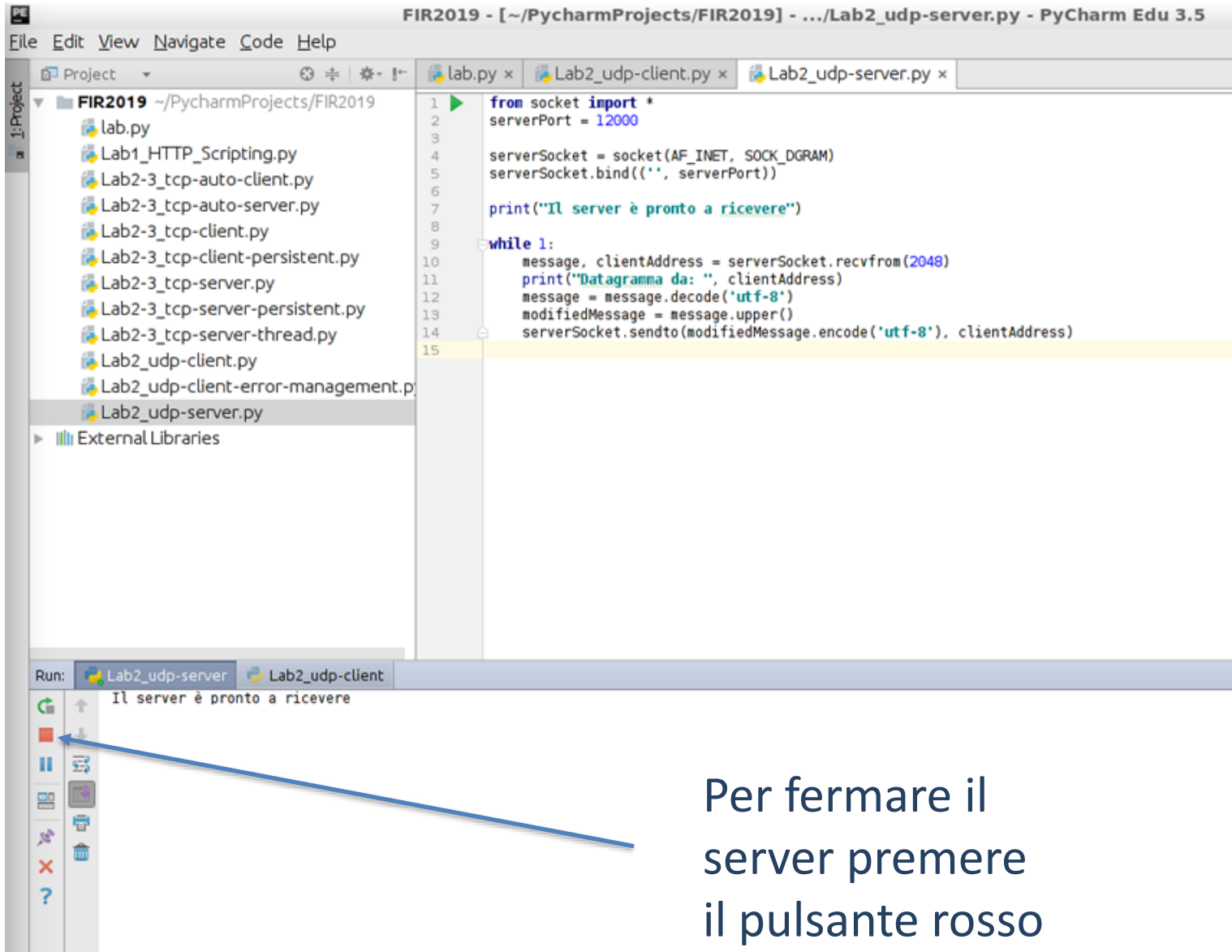
Run: Lab2_udp-server Lab2_udp-client

Inserisci lettere: *ciao*
CIAO

Process finished with exit code 0

Output

Stop del Server



Per fermare il
server premere
il pulsante rosso

Passi successivi

- Il server è ancora in esecuzione
- Il client può essere eseguito altre volte e il sistema operativo assegna una porta diversa ogni volta.

Esercizio 2.1

a) Modificare il server in modo che scriva l'indirizzo IP e numero di porta del client.

Esercizio 2.1

The screenshot shows the PyCharm IDE interface. The top toolbar includes File, Edit, View, Navigate, Code, and Help. The project browser on the left shows the project structure for FIR2019, with files like lab.py, Lab1_HTTP_Scripting.py, and various Lab2 files. The main editor displays the code for Lab2_udp-server.py. The code is as follows:

```
1 from socket import *
2 serverPort = 12000
3
4 serverSocket = socket(AF_INET, SOCK_DGRAM)
5 serverSocket.bind(('', serverPort))
6
7 print("Il server è pronto a ricevere")
8
9 while 1:
10     message, clientAddress = serverSocket.recvfrom(2048)
11     print('Datagramma da: ', clientAddress)
12     message = message.decode('utf-8')
13     modifiedMessage = message.upper()
14     serverSocket.sendto(modifiedMessage.encode('utf-8'), clientAddress)
15
```

At the bottom, the Run console shows the output of the Lab2_udp-server process:

```
Run: Lab2_udp-server Lab2_udp-client
Il server è pronto a ricevere
Datagramma da: ('127.0.0.1', 38154)
```

Two blue arrows point from the text annotations to the output. One arrow points to the IP address '127.0.0.1' and the other points to the port number '38154'.

indirizzo IP del client
127.0.0.1 = loopback

porte UDP sorgente del client
(per ogni riga un'esecuzione
del client)

Gestione degli errori: Problematiche

- Vediamo cosa accade se inviamo dati (e aspettiamo dati) da un server inesistente.
- Nel client modifichiamo la porta destinazione in 12001 (nessun processo è in ascolto su tale porta).
- Di norma il server risponde con un messaggio di errore, che deve essere gestito dal client.
- Tuttavia il messaggio di errore potrebbe perdersi oppure arrivare quando il client è bloccato sulla `socket.recvfrom`. In tal caso il messaggio viene ignorato.

Gestione degli errori: Soluzioni

Due possibili esiti:

- 1) il messaggio di errore dal server diventa un'eccezione Python
- 2) il client aspetta all'infinito un messaggio dal server

Soluzione:

- ✓ impostare un timeout alle operazioni sui socket
- ✓ catturare le eccezioni Python
 - scadenza timeout
 - messaggi di errore

Impostare un timeout

- Si usa il comando:
 - `clientSocket.setTimeout(<timeout>)`
- Può essere inserito ovunque prima della chiamata a `recvfrom`.
- Il timeout è un valore con virgola espresso in secondi.

Catturare le eccezioni: Try - Except

- Gli errori di socket (compresi i timeout) lanciano eccezioni python.
- Per catturare l'eccezione racchiudere il codice che può lanciare l'eccezione in un blocco ***try ... except***

```
try:
    modifiedMessage, serverAddress =
clientSocket.recvfrom(2048)
    print(modifiedMessage.decode('utf-8'))
except:
    print("Timeout scaduti: Server non
raggiungibile")
finally:
    clientSocket.close()
```

Esercizio 2.2

- a) Scrivere un nuovo client UDP che non dia errore nel caso di server inesistente.

- b) Scrivere un client UDP errato che termina prima di ricevere la risposta dal server. Cosa accade al server? Che modifiche sono necessarie?

Soluzione 2.2

```
from socket import *
serverName = 'localhost'
serverPort = 12001
clientSocket = socket(AF_INET, SOCK_DGRAM)
clientSocket.settimeout(2)
message = input('Inserisci lettere:')
clientSocket.sendto(message.encode('utf-8'), (serverName, serverPort))
try:
    modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
    # in case of error blocks forever
    print(modifiedMessage.decode('utf-8'))
except:
    print("Timeout scaduto: Server non raggiungibile ")
finally:
    clientSocket.close()
```

Esercizio 2.3

Si vuole scrivere un'applicazione client/server UDP per conteggiare il numero di consonanti presenti in una stringa.

- **Il client chiede all'utente di inserire una stringa**
- **il server risponde indicando il numero di consonanti presenti nella stringa (sia maiuscole che minuscole).**

Hint: `y.count(x)` conta quante volte appare l'elemento `x` nella lista `y`.

Scrivere gli script "UDP client" e "UDP server" date le seguenti specifiche:

- Utilizzare indirizzi IPv4
- Time-out in ricezione (lato client): 5 secondi
- Lunghezza buffer di ricezione: 2048 byte

Soluzione 2.3

UDP server

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))

print('Server pronto a ricevere!')
vocali = ['A','E','I','O','U']

while 1:
    messaggio, clientAddress = serverSocket.recvfrom(2048)
    messaggio = messaggio.decode('utf-8')
    num = len(messaggio)
    for voc in vocali:
        num = num - messaggio.count(voc)
    risposta = "Il messaggio contiene "+str(num)+" consonanti."
    serverSocket.sendto(risposta.encode('utf-8'), clientAddress)
```


Soluzione 2.3

UDP client

```
from socket import *
serverName = 'localhost'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_DGRAM)
clientSocket.settimeout(5)

message = input('Inserisci una parola (senza caratteri speciali):')
clientSocket.sendto(message.encode('utf-8'), (serverName,
serverPort))

try:
    reply, serverAddress = clientSocket.recvfrom(2048)
    print(reply.decode('utf-8'))
except:
    print("Il server non ha risposto entro il timeout...")
finally:
    clientSocket.close()
```

Esercizio 2.4 (Soluzione Dopo Il Lab)

Si vuole scrivere un'applicazione client/server UDP

- Il client chiede all'utente di inserire un numero**
- Il server risponde indicando se il numero inserito e' un numero primo o no**

Scrivere gli script "UDP client" e "UDP server" date le seguenti specifiche:

- Utilizzare indirizzi IPv4
- Time-out in ricezione (lato client): 2 secondi