

Reti di Calcolatori

Prof. Roberto Canonico

Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione

Corso di Laurea in Ingegneria Informatica

Esempi di programmi client/server in Python 3:
comunicazione con TCP e con UDP

I lucidi presentati al corso sono uno strumento didattico
che NON sostituisce i testi indicati nel programma del corso

Nota di copyright per le slide COMICS

Nota di Copyright

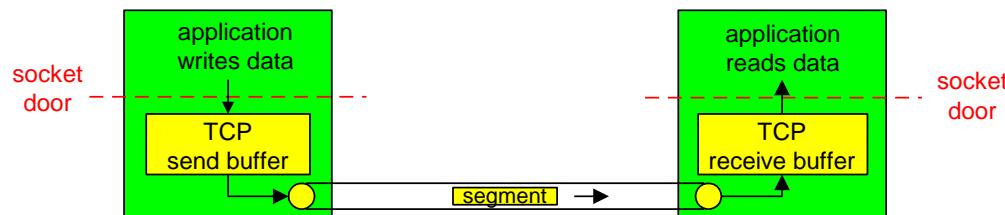
Questo insieme di trasparenze è stato ideato e realizzato dai ricercatori del Gruppo di Ricerca COMICS del Dipartimento di Informatica e Sistemistica dell'Università di Napoli Federico II. Esse possono essere impiegate liberamente per fini didattici esclusivamente senza fini di lucro, a meno di un esplicito consenso scritto degli Autori. Nell'uso dovranno essere esplicitamente riportati la fonte e gli Autori. Gli Autori non sono responsabili per eventuali imprecisioni contenute in tali trasparenze né per eventuali problemi, danni o malfunzionamenti derivanti dal loro uso o applicazione.

Autori:

Simon Pietro Romano, Antonio Pescapè, Stefano Avallone,
Marcello Esposito, Roberto Canonica, Giorgio Ventre

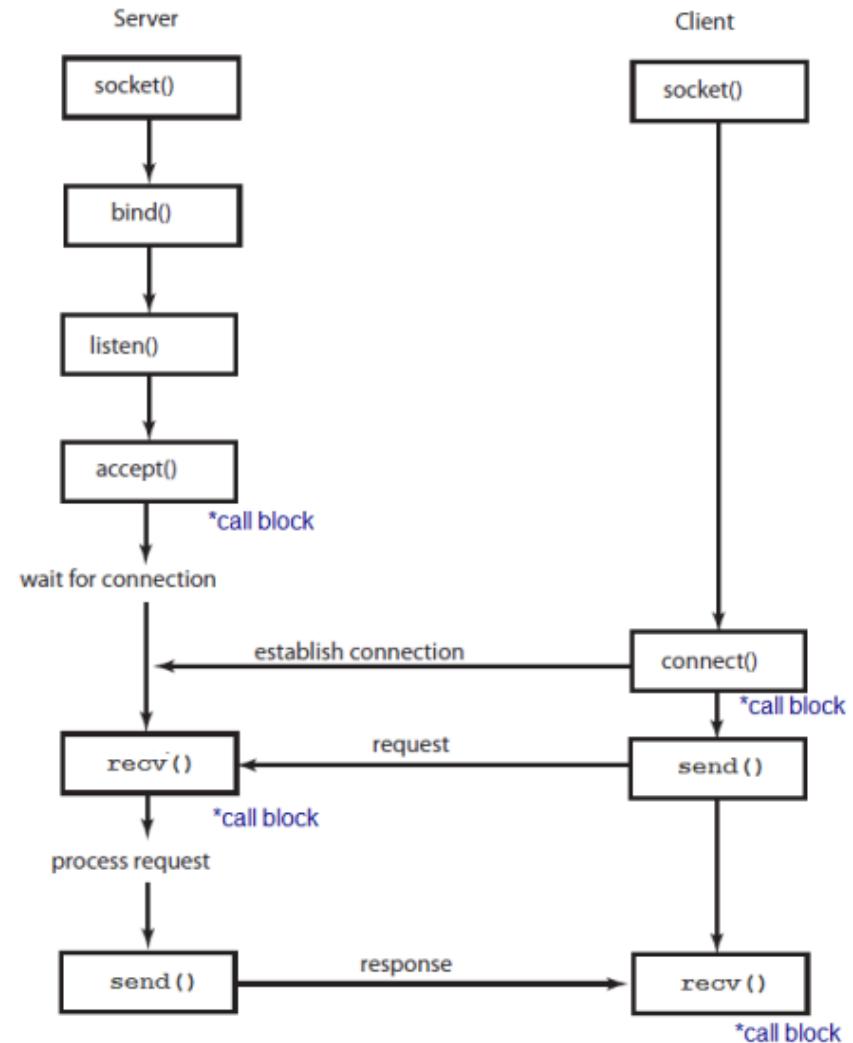
Comunicazione con TCP

- La comunicazione con TCP è *connection-oriented*
- Una volta stabilita la connessione tra due endpoint, TCP realizza il trasferimento affidabile di stream di byte in entrambi i versi
- Il processo che prende l'iniziativa di stabilire la connessione è il ***client***, l'altro è il ***server***
- Ciascun datagramma IP è associato alla connessione TCP da:
 - *Indirizzo IP* dell'host mittente
 - *Indirizzo IP* dell'host destinazione
 - *Port number sorgente* scelto dal mittente
 - *Port number destinazione* sul quale il processo destinatario è in ricezione
- In momenti diversi sia il client che il server possono inviare dati all'altra parte
- Il pattern della comunicazione dipende dal protocollo di livelli applicativo

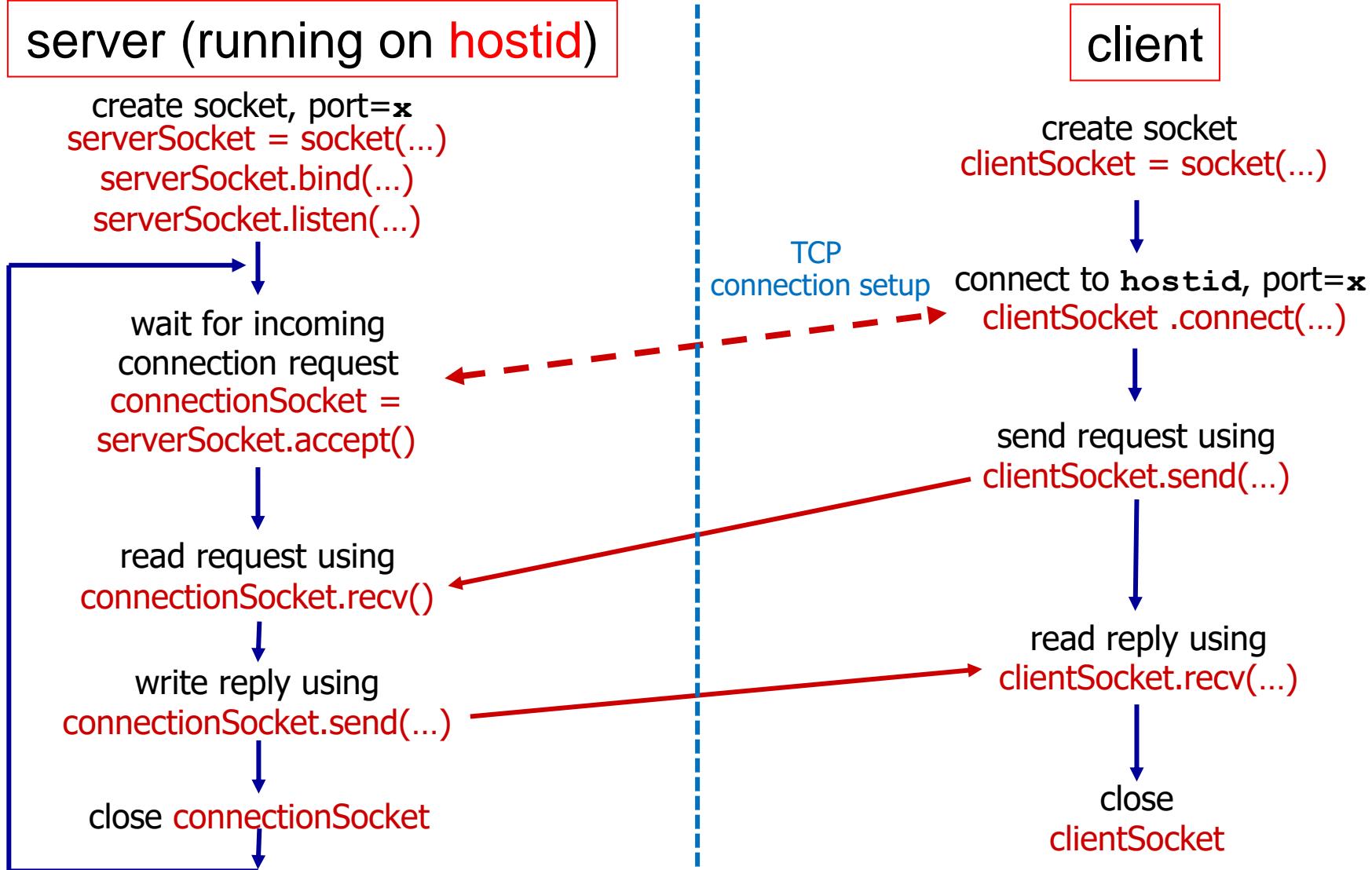


TCP in Python: client e server

- Client e server creano un oggetto di classe socket
 - Si passano come parametri le costanti predefinite `socket.AF_INET` e `socket.SOCK_STREAM`
 - Per IPv6 si usa come primo parametro il valore `socket.AF_INET6`
- Successivamente, sugli oggetti socket si invocano dei metodi secondo uno schema differente per le due entità
- Le funzioni `connect()`, `accept()` e `recv()` sono *bloccanti*
- All'uscita da `connect()` e da `accept()` la connessione è stata stabilita
 - *three-way handshake* effettuato



Client/server con TCP: schema



Python: send() vs sendall()

- Il metodo `send(bytes_to_send)` accetta come parametro un oggetto di tipo `bytes` e restituisce come valore di ritorno il numero di byte effettivamente trasferiti
 - Se il numero di byte trasferiti è inferiore alla dimensione di `bytes_to_send`, l'applicazione deve effettuare una ulteriore trasmissione dei byte residui
 - Per inviare una stringa di testo, occorre trasformare la stringa in un oggetto di tipo `bytes` usando la funzione `encode('ascii')` ed effettuare `decode('ascii')` in ricezione
- Il metodo `sendall(bytes_to_send)` accetta come parametro un oggetto di tipo `bytes`
 - La funzione `sendall()` è bloccante: si esce dalla funzione solo quando tutti i byte sono stati effettivamente trasferiti alla controparte
 - Per inviare una stringa di testo, occorre trasformare la stringa in un oggetto di tipo `bytes` usando la funzione `encode('ascii')` ed effettuare `decode('ascii')` in ricezione
 - Una funzione che si comporta come il metodo `sendall()`:

```
def sendall(sock, data, flags=0):
    ret = sock.send(data, flags)
    if ret > 0:
        return sendall(sock, data[ret:], flags)
    else:
        return None
```

TCP client in Python 3 (1/3)

```
import sys, getopt, socket
# ... def usage():
# ... def read_args():
SERVER_ADDRESS = "127.0.0.1"
SERVER_PORT = 12000
# read command line arguments
read_args()
# get user keyboard input
sentence = input("Input lowercase sentence: ")
# create TCP socket
clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# connect socket to remote server
clientSocket.connect((SERVER_ADDRESS, SERVER_PORT))
print("Connected to server at (%s, %d)" % (SERVER_ADDRESS, SERVER_PORT))
# Send sentence into socket, no need to specify server IP and port
request = sentence.encode()
clientSocket.send(request)
# read reply message from socket into response
response = clientSocket.recv(1024)
# print out received response
print("Received reply <%s>" % response.decode())
# close socket
clientSocket.close()
print("Connection closed with server at (%s, %d)" % (SERVER_ADDRESS, SERVER_PORT))
```

TCP client in Python 3 (2/3)

```
# ...

def usage():
    print("Usage: %s [-a X.X.X.X] [-p N]" % sys.argv[0])
    print("Arguments:")
    print("-a X.X.X.X --> server listening address (default 127.0.0.1)")
    print("-p N   --> server listening port number in 1024..65535 (default 12000)")
    print("Flags:")
    print("-h      --> help")

# ...
```

TCP client in Python 3 (3/3)

```
# ...

def read_args():
    global SERVER_ADDRESS, SERVER_PORT
    try:
        opts, args= getopt.getopt(sys.argv[1:], 'ha:p:', ['help', 'address=', 'port='])
    except getopt.GetoptError as err:
        print("ERROR: ", err)
        usage(); sys.exit()

    for opt, arg in opts:
        if opt in ('-h', '--help'):
            usage(); sys.exit()
        elif opt in ('-a', '--address'):
            SERVER_ADDRESS = arg
        elif opt in ('-p', '--port'):
            if (arg.isnumeric()) and (int(arg) >= 1024) and (int(arg) <= 65535):
                SERVER_PORT = int(arg)
            else:
                print("ERROR: -p option must be an integer in 1024..65535")
                usage(); sys.exit()
        else:
            print("Unrecognized argument !")
            usage(); sys.exit()

# ...
```

TCP server in Python 3

```
import sys, getopt, socket
# ... def usage():
# ... def read_args():
SERVER_ADDRESS = "0.0.0.0"
SERVER_PORT = 12000
# read command line arguments
read_args()
# create TCP socket
serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serverSocket.bind((SERVER_ADDRESS, SERVER_PORT)) # bind socket to local port
serverSocket.listen(1) # put socket in passive mode
while True:
    # wait for incoming connections on accept(), new socket created on return
    print("Server waiting on (%s, %d)" % (SERVER_ADDRESS, SERVER_PORT))
    connectionSocket, clientAddress = serverSocket.accept()
    # receive request on newly established connectionSocket
    request = connectionSocket.recv(1024)
    print("Received request from (%s, %d)" % (clientAddress[0], clientAddress[1]))
    # convert message to upper case
    response = request.upper()
    # send back modified string to client
    connectionSocket.send(response)
    print("Sent reply to (%s, %d)" % (clientAddress[0], clientAddress[1]))
    connectionSocket.close()
```

HTTP client in Python 3 (1/2)

```
import socket
from urllib.parse import urlparse

while True:
    url = input("Type the object's URL: ")
    if (url == ""):
        break
    elif (url == "*"):
        url = "http://127.0.0.1:8080/"

    o = urlparse(url)
    if (o.hostname != None):
        SERVER_ADDRESS = o.hostname
    else:
        print("Error in parsing URL <%s>. Retry." % url); continue

    if (o.port != None):
        SERVER_PORT = o.port
    else:
        SERVER_PORT = 80

    if (o.path != None):
        PATH = o.path
    else:
        PATH = "/"

    #...
```

HTTP client in Python 3 (2/2)

```
#...
clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print("Connecting to server at (%s, %d) " % (SERVER_ADDRESS, SERVER_PORT))
try:
    clientSocket.connect((SERVER_ADDRESS, SERVER_PORT))
except OSError as err:
    print("Error. Retry."); continue

request = "GET %s HTTP/1.0\r\n\r\n" % PATH
print(request)
clientSocket.send(request.encode('ascii'))

print("Waiting for response ", end="... ")
reply = clientSocket.recv(1500)
print("received")
clientSocket.close()

print("Response:")
print(reply.decode('ascii'))
```

HTTP server in Python 3

```
import http.server
import socketserver

Handler = http.server.SimpleHTTPRequestHandler
DEFAULT_PORT = 8080

try:
    port = int(sys.argv[1])
except:
    port = DEFAULT_PORT

print ("Serving at port %s/tcp ..." % port)

httpd = socketserver.TCPServer(("0.0.0.0", port), Handler)
httpd.serve_forever()...
```

Client/server con UDP: schema

server (running on hostid)

create socket, port= **x**:

```
serverSocket = socket(AF_INET,SOCK_DGRAM)
```

read datagram from **serverSocket**

write reply to **serverSocket**
specifying
client address, port number

client

create socket:

```
clientSocket = socket(AF_INET,SOCK_DGRAM)
```

Create datagram with **hostid** and port=**x**;
send datagram via **clientSocket**
to hostid, port **x**

read datagram from **clientSocket**

close **clientSocket**

UDP client in Python 3

```
import sys, getopt, socket
# ... def usage():
# ... def read_args():
SERVER_ADDRESS = "127.0.0.1"
SERVER_PORT = 12000
# read command line arguments
read_args()
# get user keyboard input
request = input("Input lowercase sentence: ")
# create UDP socket
clientSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# send request to server
clientSocket.sendto(request.encode(), (SERVER_ADDRESS, SERVER_PORT))

# wait for server response
print("Waiting for server response...")
response, serverAddress = clientSocket.recvfrom(2048)

# Print out received response
print(response.decode())

# Close socket
clientSocket.close()
```

UDP server in Python 3

```
import sys, getopt, socket
# ... def usage():
# ... def read_args():
SERVER_ADDRESS = "0.0.0.0"
SERVER_PORT = 12000
# read command line arguments
read_args()
# create UDP socket
serverSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
serverSocket.bind((SERVER_ADDRESS, SERVER_PORT)) # bind socket to local port

while True:
    print("Server waiting on (%s, %d)" % (SERVER_ADDRESS, SERVER_PORT))
    # receive request
    request, clientAddress = serverSocket.recvfrom(2048)
    print("Received request from (%s, %d)" % (clientAddress[0], clientAddress[1]))
    # convert message to upper case
    response = request.upper()
    # send back modified string to client
    serverSocket.sendto(response, clientAddress)
    print("Sent reply to (%s, %d)" % (clientAddress[0], clientAddress[1]))
```