SLIDES DURANTE IL LABORATORIO

Fondamenti di Comunicaizoni e Internet

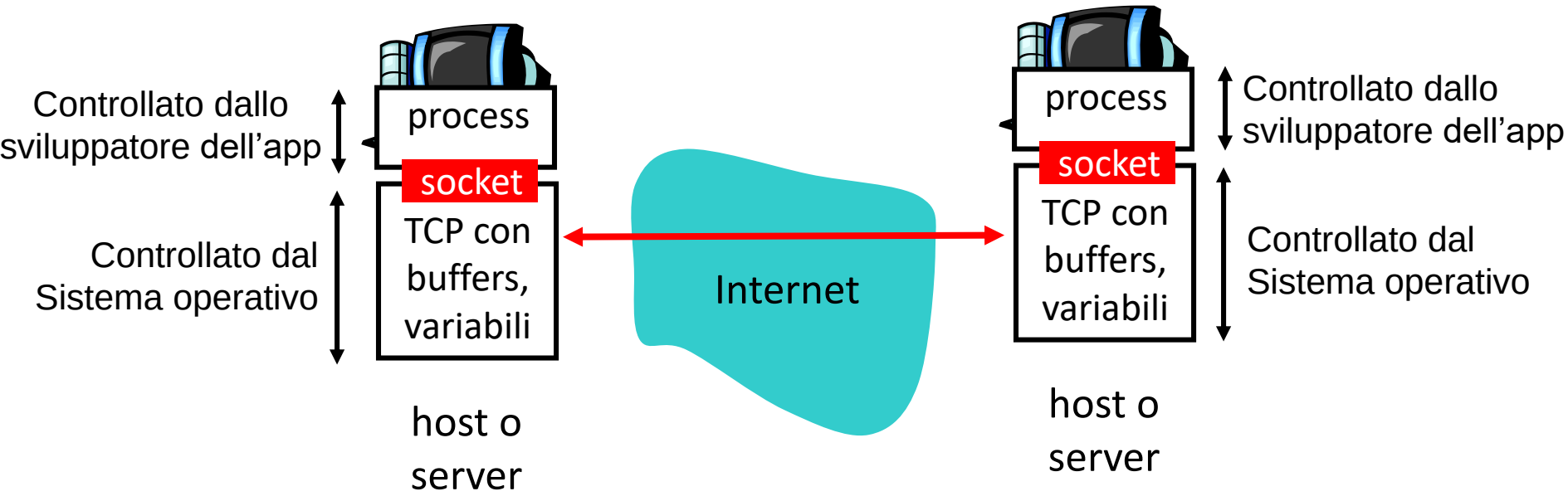
Antonio Capone, Matteo Cesana, Guido Maier, Francesco Musumeci

Programmazione Socket

Antonio Capone, Matteo Cesana,
Guido Maier, Francesco Musumeci

Programmazione socket con TCP

Servizio TCP: trasferimento affidabile di **bytes** da un processo all'altro



Programmazione socket con TCP

1. Attivazione Processo Server:

Il processo server deve essere eseguito **per primo** e deve aver **creato un socket (porta)** per accogliere le richieste del client

Welcome socket: assomiglia al socket UDP e può ricevere da tutti i client

2. Il Client deve contattare il Server:

Creando un socket TCP e

specificando indirizzo IP e numero di porta del processo server

Punto di vista dell'applicazione

TCP fornisce trasferimento affidabile e ordinato di bytes ("pipe") tra client eserver

Programmazione socket con TCP

3. Instaurazione della connessione:

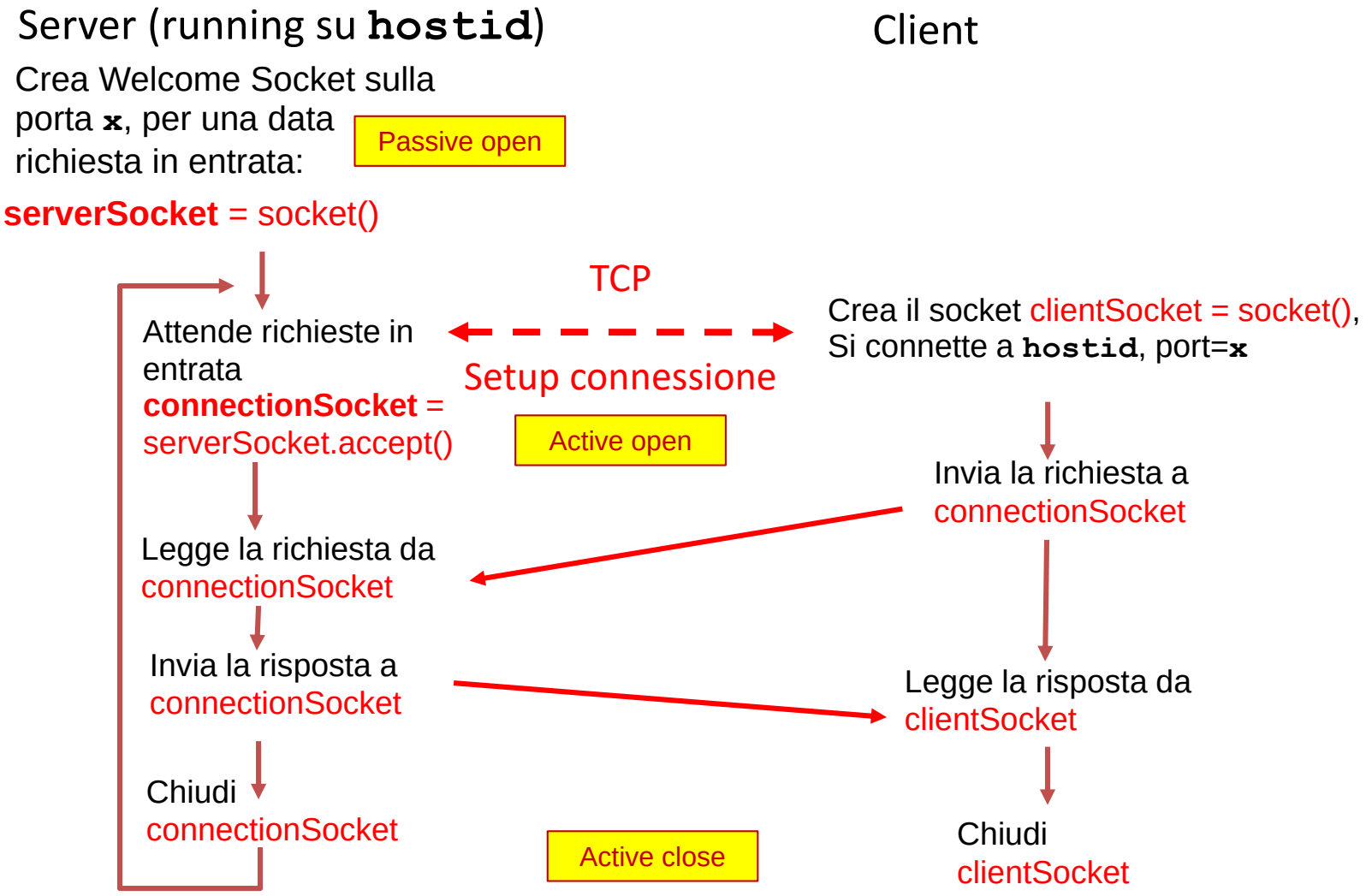
Il Client TCP **instaura una connessione** tra il proprio socket e il welcome socket del Server TCP

4. Inizio della trasmissione:

Quando contattato dal client, il **server TCP crea un nuovo socket per la comunicazione tra processo server e client:**

- Così da poter **comunicare con diversi client**
- **Usando i numeri di porta sorgente per distinguere i client**

Interazione tra socket Client/server: TCP



Programmazione socket con TCP

Esempio applicazione client-server:

1) Il client legge una riga dallo standard input (**`inFromUser`** stream) e la invia al server attraverso un socket (**`outToServer`** stream)

2) Il server legge una riga dal socket

3) Il server converte la riga in maiuscola e le invia al client

4) Il client legge la riga dal socket (**`inFromServer`** stream) e la mostra all'utente

Applicazione d’esempio: TCP client

```
from socket import *
serverName = [servername]
serverPort = 12000
```

Nome simbolico del server

Crea un socket TCP verso il server remoto, porta 12000

Applicazione d'esempio: TCP client

```
from socket import *
serverName = [servername]
serverPort = 12000

clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))
```

Nome simbolico del server

Crea un socket TCP verso il server remoto, porta 12000

SOCK_STREAM → TCP

Server port #

Applicazione d'esempio: TCP client

```
from socket import *
serverName = [servername]
serverPort = 12000

clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))

message = input('Inserisci lettere:')
clientSocket.send(message.encode('utf-8'))
```

Nome simbolico del server

Crea un socket TCP verso il server remoto, porta 12000

SOCK_STREAM → TCP

Server port #

Non è necessario appendere indirizzo e porta del server

Applicazione d'esempio: TCP client

```
from socket import *
serverName = [servername]
serverPort = 12000

clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))

message = input('Inserisci lettere:')
clientSocket.send(message.encode('utf-8'))

modifiedMessage = clientSocket.recv(1024)
Print('Dal Server:', modifiedMessage.decode('utf-8'))
clientSocket.close()
```

Nome simbolico del server

Crea un socket TCP verso il server remoto, porta 12000

SOCK_STREAM → TCP

Server port #

Non è necessario appendere indirizzo e porta del server

Dimensione del buffer

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

Applicazione d’esempio: TCP server

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind((‘’,serverPort))
serverSocket.listen(1)
```

Crea welcoming socket TCP

Il server si mette in ascolto per connessioni TCP in ingresso

Socket di ascolto (passive open)

Applicazione d'esempio: TCP server

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind((‘’,serverPort))
serverSocket.listen(1)
Print(‘Il server è pronto a ricevere’)
while 1:
    connectionSocket, clientAddress = serverSocket.accept()
    print(‘Connesso con: ‘, clientAddress)
```

Crea welcoming socket TCP

Il server si mette in ascolto per connessioni TCP in ingresso

Socket di ascolto (passive open)

loop infinito

Il server resta in attesa di richieste di connessione sull' accept();
a connessione instaurata viene ritornato un nuovo socket

All'arrivo di una active open (accept) la server socket crea una connection socket dedicata a quel client

Applicazione d'esempio: TCP server

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind((‘’,serverPort))
serverSocket.listen(1)
Print(‘Il server è pronto a ricevere’)
while 1:
    connectionSocket, addr = serverSocket.accept()
    print(‘Connesso con: ‘, clientAddress)
    message = connectionSocket.recv(1024)
    message = message.decode(‘utf-8’)
    modifiedMessage = message.upper()
```

Crea welcoming socket TCP

Il server si mette in ascolto per connessioni TCP in ingresso

Socket di ascolto (passive open)

loop infinito

Il server resta in attesa di richieste di connessione sull' accept(); a connessione instaurata viene ritornato un nuovo socket

All'arrivo di una active open (accept) la server socket crea una connection socket dedicata a quel client

Legge bytes dal socket (non legge anche l'indirizzo, come in UDP)

Applicazione d'esempio: TCP server

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind((‘’,serverPort))
serverSocket.listen(1)
Print(‘Il server è pronto a ricevere’)
while 1:
    connectionSocket, addr = serverSocket.accept()
    print(‘Connesso con: ‘, clientAddress)
    message = connectionSocket.recv(1024)
    message = message.decode(‘utf-8’)
    modifiedMessage = message.upper()
    connectionSocket.send(modifiedMessage.encode(‘utf-8))
    connectionSocket.close()
```

Crea welcoming socket TCP

Il server si mette in ascolto per connessioni TCP in ingresso

Socket di ascolto (passive open)

loop infinito

Il server resta in attesa di richieste di connessione sull' accept(); a connessione instaurata viene ritornato un nuovo socket

All'arrivo di una active open (accept) la server socket crea una connection socket dedicata a quel client

Legge bytes dal socket (non legge anche l'indirizzo, come in UDP)

Chiude la connessione col client corrente (ma *non* il welcoming socket!)

Esercizio 3.1

Si vuole scrivere un'applicazione client/server TCP 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 "TCP client" e "TCP server" date le seguenti specifiche:

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

Soluzione 3.1

TCP server

```
1   from socket import *
2   serverPort = 12000
3
4   serverSocket = socket(AF_INET, SOCK_STREAM)
5   serverSocket.bind(('',serverPort))
6
7   serverSocket.listen(1)
8   print('Server Pronto!')
9   vocali = ['A','E','I','O','U','a','e','i','o','u']
10
11  while 1:
12      connectionSocket, clientAddress = serverSocket.accept()
13      print(clientAddress)
14      message = connectionSocket.recv(2048)
15      message = message.decode('utf-8')
16      numero = len(message)
17      for voc in vocali:
18          numero = numero - message.count(voc)
19      risposta = 'Il numero di consonanti è: '+ str(numero)
20      connectionSocket.send(risposta.encode('utf-8'))
21      connectionSocket.close()
```

Soluzione 3.1

TCP client

```
1    from socket import*
2
3    serverName = 'localhost'
4    serverPort = 12001
5
6    clientSocket = socket(AF_INET, SOCK_STREAM)
7    clientSocket.connect((serverName,serverPort))
8
9    clientSocket.settimeout(2)
10
11   message = input('Per favore inserire lettere:')
12   clientSocket.send(message.encode('utf-8'))
13
14   try:
15       modifiedmessage, serverAddress = clientSocket.recvfrom(2048)
16       modifiedmessage = modifiedmessage.decode('utf-8')
17       print(modifiedmessage)
18   except:
19       print('Server non raggiugibile, prova piu tardi')
20   finally:
21       clientSocket.close()
```

Connessioni persistenti

Una coppia client e server TCP possono usare un socket aperto più volte.

TCP non fornisce un meccanismo di delimitazione dei messaggi applicativi. Quindi un protocollo applicativo deve determinare:

- quando iniziano/finiscono i messaggi
- quando terminare la connessione

Connessioni persistenti - Esempio

Scriviamo un'applicazione TCP client-server tale che:

- Il client invia una stringa di caratteri al server
- Il server risponde con la stessa string in cui le lettere minuscole sono rese maiuscole

Nel nostro esempio ogni messaggio è terminato da un carattere «a capo» e la connessione va terminata quando il messaggio è un singolo punto «.»

Client persistente

```
1   from socket import *
2
3   serverName = 'localhost'
4   serverPort = 12000
5
6   clientSocket = socket(AF_INET, SOCK_STREAM)
7   clientSocket.connect((serverName, serverPort))
8
9   while True:
10      sentence = input('Inserisci lettere ( . per fermare):')
11      clientSocket.send(sentence.encode('utf-8'))
12      if sentence == '.':
13          break
14      modifiedSentence = clientSocket.recv(1024)
15      print('Dal Server:', modifiedSentence.decode('utf-8'))
16
17  clientSocket.close()
```

Server persistente

```
1  from socket import *
2
3
4  serverPort = 12000
5  serverSocket = socket(AF_INET, SOCK_STREAM)
6
7  serverSocket.bind(('', serverPort))
8  serverSocket.listen(1)
9
10 while True:
11     print('Il server è pronto a ricevere')
12     connectionSocket,  clientAddress = serverSocket.accept()
13     print("Connesso con: ", clientAddress)
14     while True:
15         sentence = connectionSocket.recv(1024)
16         if sentence.decode('utf-8') == '.':
17             break
18         capitalizedSentence = (sentence.decode('utf-8')).upper()
19         connectionSocket.send(capitalizedSentence.encode('utf-8'))
20     connectionSocket.close()
```

Esercizio 3.2

a) Eseguire il client persistente e inviare messaggi multipli
b) Senza chiudere il primo client, aprirne un secondo e poi un terzo. Cosa accade?
c) Inviare un messaggio con il secondo client. Cosa accade?
d) Terminare il primo client. Cosa accade ora al secondo client?

Coda delle connessioni incomplete

Le richieste di apertura di connessione si accodano in una coda di connessioni incomplete. Il comando **accept** preleva la prima connessione incompleta e crea un socket.

Con il comando:

```
serverSocket.listen(1)
```

abbiamo istanziato una coda delle connessioni incomplete di lunghezza 1.

Quando il server sta servendo il primo client, la seconda richiesta rimane in coda, mentre la terza viene rifiutata.

Esercizio 3.3

Si vuole scrivere un'applicazione client/server TCP, tale che:

- **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 "TCP client" e "TCP server" date le seguenti specifiche:

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

Soluzione 3.3 - Server

```
1 from socket import *
2
3 def prime_checker(num):
4     # I numeri primi sono maggiori di 0
5     try:
6         num = int(num)
7     except:
8         prime_flag = -1
9         return str(prime_flag)
10    if num > 1:
11        prime_flag = 1
12        for i in range(2, num): # Cerca i Fattori
13            if (num % i) == 0:
14                prime_flag = 0  # "num non è un numero primo"
15                break
16    else: # entriamo in questo blocco se num è minore di 1 (1 è considerato NON primo)
17        prime_flag = 0 # "num non è un numero primo"
18    return str(prime_flag)
19 serverPort = 12000
20 serverSocket = socket(AF_INET, SOCK_STREAM)
21 serverSocket.bind(('', serverPort))
22 serverSocket.listen()
23 print("Il server è pronto a ricevere")
24 while 1:
25     connectionSocket, clientAddress = serverSocket.accept()
26     message = connectionSocket.recvfrom(2048)
27     print("Datagramma da: ", clientAddress)
28     message = message.decode('utf-8')
29     isprime = prime_checker(message)
30     connectionSocket.send(isprime.encode('utf-8'))
31     connectionSocket.close()
```

Soluzione 3.3 - Client

```python
from socket import *

serverName = 'localhost'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))

clientSocket.settimeout(10) # Fissa un timeout di 2 [s] per Server non raggiungibili

message = input('Inserisci un numero:')
clientSocket.send(message.encode('utf-8'))

try:
    isprime, serverAddress = clientSocket.recvfrom(2048)
    isprime = isprime.decode('utf-8')
    if isprime == "1":
        print('Il numero inserito è un numero primo.')
    elif isprime == "0":
        print('Il numero inserito NON è un numero primo.')
    else:
        print('Il carattere inserito non è corretto.\nChiusura Sessione.')
except: # Entriamo in questo blocco se il Server non risponde entro il timeout
    print("Timeout scaduto: Server non raggiungibile ")
finally:
    clientSocket.close()

```

Multithreaded servers

Per gestire client multipli il server deve attivare **più worker**:

- un worker per gestire le richieste di connessione entranti (listening socket)
- un worker per gestire ciascuna connessione (active socket)

Multithreaded servers

Per gestire client multipli il server deve attivare **più worker**:

- un worker per gestire le richieste di connessione entranti (listening socket)
- un worker per gestire ciascuna connessione (active socket)

Tre paradigmi:

1. ogni worker è un thread
    - facile e veloce
    - alcune limitazioni (numero max di thread paralleli, tutti i thread hanno gli stessi permessi etc…)
2. ogni worker è un processo
    - aggiunge complessità e latenza
3. un solo worker che serve a turno un messaggio da ogni socket (molto efficiente, ma più complicato da gestire).

Multithreaded servers

```
1   from socket import *
2   from threading import Thread
3
4
5   def handler(connectionSocket):
6       while True:
7           sentence = connectionSocket.recv(1024)
8           if sentence.decode('utf-8') == '.':
9               break
10          capitalizedSentence = (sentence.decode('utf-8')).upper()
11          connectionSocket.send(capitalizedSentence.encode('utf-8'))
12      connectionSocket.close()
13
14
15  serverPort = 12000
16  serverSocket = socket(AF_INET, SOCK_STREAM)
17  serverSocket.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
18  serverSocket.bind(('', serverPort))
19  serverSocket.listen(1)
20
21  while True:
22      print('Il server è pronto a ricevere')
23      newSocket,  addr = serverSocket.accept()
24      thread = Thread(target=handler, args=(newSocket,))
25      thread.start()
```

Il codice per gestire ogni connessione attiva va in una funzione

Bisogna consentire al socket locale di usare più volte la stessa porta

Ogni connessione viene servita in un nuovo thread