**Prova in itinere – 4 Maggio 2017**

|  |  |
| --- | --- |
| **Cognome** | **STUDENTE** |
| **Nome** | **BRAVO** |
| **Matricola** | **SOLUZIONI** |

##### Tempo complessivo a disposizione per lo svolgimento: 2h

**Usare lo spazio dopo ogni Esercizio/Quesito per la risposta**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Es1 (5 pt)** | **Es2 (6 pt)** | **Es3 (6 pt)** | **Ques (9 pt)** | **Lab (6pt)** |
|  |  |  |  |  |

## 1- Esercizio (5 punti)

In una rete a commutazione di pacchetto al tempo t=0 sono presenti 6 pacchetti in H diretti rispettivamente alle seguenti destinazioni: B, B, D, D, C, C. Calcolare il tempo di ricezione di ciascuno dei pacchetti, a partire da t=0, assumendo che i pacchetti abbiamo le seguenti dimensioni: LB=1250 Byte, LC=250 Byte, LD=1250 Byte.



SOLUZIONE

Tempi di trasmissione sui link

## 





## 2 - Esercizio (6 punti)

Nella stessa rete dell’es. 1, una connessione TCP tra l’*host* A e l’*host* H è caratterizzata dai seguenti parametri: lunghezze di *header, ack* e *segmenti di apertura* trascurabili, *link* bidirezionali simmetrici, *MSS* = 1250 Byte, *RCWND* >> *CWND, SSTHRESH* = 10000 Byte.

1. Si calcoli la lunghezza della finestra che permette la trasmissione continua Wc
2. Si calcoli il tempo necessario (da prima dell’apertura della connessione alla ricezione dell’ultimo ACK) a trasferire un file di 50 KByte dall’*host* A all’*host* H.
3. Si ripeta il calcolo assumendo che il 34° segmento vada perso e il timeout corrispondente sia Tout=100 ms (si assuma i pacchetti fuori sequenza NON vengano memorizzati)

SOLUZIONE



## 3 - Esercizio (6 punti)

Si assuma H sia un server HTTP e A un client HTTP. Occorre trasferire un documento HTML base di 125 KByte e 10 immagini di 1.25 MByte in presenza di 5 flussi interferenti tra F e E, e 4 flussi interferenti tra G ed E.

Assumendo:

* un RTT = 46.5 ms per i messaggi HTTP tra A e H,
* il tempo totale di apertura della connessione TCP, Topen = 40 ms,
* un ritmo medio di trasmissione pari al valore di condivisione equa delle risorse e
* RTT e Topen indipendenti dal ritmo medio di trasferimento

si calcoli il tempo di trasferimento necessario nel caso di

1. connessione HTTP persistente (una singola connessione) e
2. connessione HTTP non persistente (con trasmissione in parallelo delle immagini).

SOLUZIONE



## Quesiti (9 punti)

**Q1)** Da un mailserver (mail.polimi.it) viene eseguito il comando dig. **A)** Quali sono i server authoritative (nome ed indirizzo IP)? **B)** Il server da cui arriva la risposta è authoritative? **C)** A quali server (nome ed indirizzo IP) potrebbe essere spedita una mail indirizzata a groot@marvel.com?

Mailserver:~ root$ dig -t ANY marvel.com

; <<>> DiG 9.8.3-P1 <<>> -t ANY marvel.com

;; global options: +cmd

;; Got answer:

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 41756

;; flags: qr rd ra; QUERY: 1, ANSWER: 8, AUTHORITY: 2, ADDITIONAL: 8

;; QUESTION SECTION:

;marvel.com. IN ANY

;; ANSWER SECTION:

marvel.com. 86188 IN MX 10 ASPMX2.GOOGLEMAIL.COM.

marvel.com. 86188 IN MX 5 ALT2.ASPMX.L.GOOGLE.COM.

marvel.com. 86188 IN MX 5 ALT1.ASPMX.L.GOOGLE.COM.

marvel.com. 86188 IN MX 10 ASPMX3.GOOGLEMAIL.COM.

marvel.com. 86188 IN MX 1 ASPMX.L.GOOGLE.COM.

marvel.com. 35392 IN A 72.32.138.96

marvel.com. 35392 IN NS ns2.rackspace.com.

marvel.com. 35392 IN NS ns.rackspace.com.

;; AUTHORITY SECTION:

marvel.com. 35392 IN NS ns.rackspace.com.

marvel.com. 35392 IN NS ns2.rackspace.com.

;; ADDITIONAL SECTION:

aspmx.l.google.com. 239 IN A 74.125.133.27

alt1.aspmx.l.google.com. 239 IN A 64.233.165.27

alt2.aspmx.l.google.com. 239 IN A 74.125.130.27

ns.rackspace.com. 39453 IN A 69.20.95.4

ns2.rackspace.com. 16426 IN A 65.61.188.4

aspmx.l.google.com. 239 IN AAAA 2a00:1450:400c:c07::1b

alt1.aspmx.l.google.com. 239 IN AAAA 2a00:1450:4010:c08::1a

alt2.aspmx.l.google.com. 239 IN AAAA 2404:6800:4003:c01::1a

;; Query time: 4 msec

;; SERVER: 62.101.93.101#53(62.101.93.101)

;; WHEN: Mon May  1 09:52:31 2017

;; MSG SIZE  rcvd: 439

A)

ns.rackspace.com, 69.20.95.4

ns2.rackspace.com, 65.61.188.4

B)

No, la risposta arriva dal server con IP 62.101.93.10

C)

aspmx.l.google.com, 74.125.133.27

alt1.aspmx.l.google.com, 64.233.165.27

alt2.aspmx.l.google.com, 74.125.130.27

**Q2)** Il server di posta della marvel si collega al server mail.polimi.it per inviare un messaggio che ha come *mittente*: Groot <groot@marvel.com> e come *destinatario*: Uno Studente <studente-fir@polimi.it>. Il messaggio ha come *subject*: Io sono Groot, e come *contenuto*: Io sono Groot.

Si indichino i messaggi SMTP inviati dal client e cosa ci si aspetta risponda il server (descrizione a parole della risposta del server).

**Client** **Server**

Welcome del server

HELO marvel

Identificazione del server

MAIL FROM: Groot <groot@marvel.com>

OK

RCPT TO: Uno Studente <studente-fir@polimi.it>.

OK

DATA

Inserisci i dati terminando con un punto su linea singola

Subject: Io sono Groot

Io sono Groot.

OK

QUIT

Bye

**Q3**

Si completi la figura in accordo alle regole del protocollo Go-back-N con N=4. Si inseriscano i valori di SN (iniziando da 0) ed RN, si indichino gli istanti di accettazione delle trame corrette e in sequenza.





## Laboratorio (6 punti)

I due script rappresentano un semplice sistema Client/Server per un sistema di voto elettronico (assumiamo per esempio un client per ogni cabina elettorale e un server centrale).

Per semplicità sono trascurate schede bianche o nulle ed i voti validi sono solo “0” e “1”

La figura riporta il protocollo applicativo utilizzato tra la componente Client e quella Server.

“LISTA”

“\*\*Elezioni Pres…”

“A”

“KO”

Client

Server

FASE 1: Il Client richiede al Server il messaggio da mostrare a video all’utente

FASE 2: Il Client invia il voto al Server.

Il server ascolta ripetutamente e risponde con “OK” /“KO” finchè l’utente non ha inviato una scelta valida

“0”

“OK”

**SCRIPT CLIENT**

from socket import \*

socket = socket(**AF\_INET, SOCK\_STREAM**)

**socket.connect(('10.0.0.1',2017))**

socket.send('LISTA')

intro\_msg = socket.recv(2048)

print intro\_msg

msg = 'KO'

while msg != 'OK':

voto = raw\_input('Inserisci preferenza [0/1]: ')

socket.**send(voto)**

msg = socket.**recv(2048)**

print msg

socket.close()

**SCRIPT SERVER**

from socket import \*

msg = "\*\*Elezioni Presidenziali Francesi 2017\*\*\n" \

"[0] Emmanuel Macron\n" \

"[1] Marine Le Pen"

socket = socket(**AF\_INET, SOCK\_STREAM**)

socket.bind(**('',2017))**)

socket.listen(2)

voti = [0, 0]

while True:

**clSocket, addr = socket.accept()**

print 'Nuovo client', addr

comando = ''

while comando != 'LISTA':

comando = clSocket.recv(2048)

clSocket.send(msg)

voto = clSocket.recv(2048)

# v.isdigit() restituisce True/False se la var. v è una stringa che rappresenta un numero

while not (voto.isdigit() and int(voto) in [0,1]):

clSocket.send('KO')

voto = clSocket.recv(2048)

clSocket.send('OK')

clSocket.close()

# implementazione incremento voti

# codice non riportato, ma

# da scrivere nella risposta di [Q2]

print 100.0\*voti[0]/sum(voti),'vs',100.0\*voti[1]/sum(voti)

memorizza\_su\_file() # implementazione del metodo omessa

socket.close()

**Q1)** Completare il codice mancante (al posto dei puntini) nel Server e nel Client per implementare l’instaurazione della connessione e la FASE 2 del protocollo, assumendo che:

* la comunicazione avvenga su TCP+IPv4
* il Server sia in ascolto all’indirizzo 10.0.0.1 sulla porta 2017
* la dimensione del buffer del socket in ricezione sia 2048 byte

**Q2)** Scrivere nello spazio sottostante il codice mancante nel Server per implementare il conteggio dei voti del candidato scelto. Hint: int(v) ritorna una versione numerica della stringa v.

voti[int(voto)] += 1

**Codice esercizi laboratorio**

**UDP client**

from socket import \*

serverName = 'localhost'

serverPort = 12000

clientSocket = socket(AF\_INET, SOCK\_DGRAM)

message = raw\_input('Input lowercase sentence:')

clientSocket.sendto(message, (serverName, serverPort))

modifiedMessage, serverAddress = clientSocket.recvfrom(2048)

print modifiedMessage

clientSocket.close()

**UDP server**

from socket import \*

serverPort = 12000

serverSocket = socket(AF\_INET, SOCK\_DGRAM)

serverSocket.bind(('', serverPort))

print "The server is ready to receive"

while 1:

message, clientAddress = serverSocket.recvfrom(2048)

print "Datagram from: ", clientAddress

modifiedMessage = message.upper()

serverSocket.sendto(modifiedMessage, clientAddress)

**UDP error management**

from socket import \*

serverName = 'localhost'

serverPort = 12001

clientSocket = socket(AF\_INET, SOCK\_DGRAM)

clientSocket.settimeout(5)

message = raw\_input('Input lowercase sentence:')

try:

clientSocket.sendto(message, (serverName, serverPort))

modifiedMessage, serverAddress = clientSocket.recvfrom(2048)

# in case of error blocks forever

print modifiedMessage

except error, v:

print "Failure"

print v

finally:

clientSocket.close()

**TCP client**

from socket import \*

serverName = 'localhost'

serverPort = 12000

clientSocket = socket(AF\_INET, SOCK\_STREAM)

clientSocket.connect((serverName, serverPort))

sentence = raw\_input('Input lowercase sentence:')

clientSocket.send(sentence)

modifiedSentence = clientSocket.recv(1024)

print 'From Server:', modifiedSentence

clientSocket.close()

**TCP server**

from socket import \*

serverPort = 12000

serverSocket = socket(AF\_INET, SOCK\_STREAM)

serverSocket.bind(('', serverPort))

serverSocket.listen(1)

print 'The server is ready to receive'

while True:

connectionSocket, clientAddress = serverSocket.accept()

print "Connection form: ", clientAddress

sentence = connectionSocket.recv(1024)

capitalizedSentence = sentence.upper()

connectionSocket.send(capitalizedSentence)

connectionSocket.close()

**TCP client persistent**

from socket import \*

serverName = 'localhost'

serverPort = 12000

clientSocket = socket(AF\_INET, SOCK\_STREAM)

clientSocket.connect((serverName, serverPort))

while True:

sentence = raw\_input('Input lowercase sentence ( . to stop):')

clientSocket.send(sentence)

if sentence == '.':

break

modifiedSentence = clientSocket.recv(1024)

print 'From Server:', modifiedSentence

clientSocket.close()

**TCP server persistent**

from socket import \*

serverPort = 12000

serverSocket = socket(AF\_INET, SOCK\_STREAM)

serverSocket.bind(('', serverPort))

serverSocket.listen(1)

while True:

print 'The server is ready to receive'

connectionSocket, clientAddress = serverSocket.accept()

print "Connection form: ", clientAddress

while True:

sentence = connectionSocket.recv(1024)

if sentence == '.':

break

capitalizedSentence = sentence.upper()

connectionSocket.send(capitalizedSentence)

connectionSocket.close()

**TCP auto client**

from socket import \*

import time

serverName = 'localhost'

serverPort = 12000

clientSocket = socket(AF\_INET, SOCK\_STREAM)

clientSocket.connect((serverName, serverPort))

for a in range(100):

clientSocket.send('A')

time.sleep(1)

clientSocket.send('.')

#clientSocket.recv(1024)

clientSocket.close()

**TCP auto server**

from socket import \*

serverPort = 12000

serverSocket = socket(AF\_INET, SOCK\_STREAM)

serverSocket.bind(('', serverPort))

serverSocket.listen(1)

while True:

print 'The server is ready to receive'

connectionSocket, clientAddress = serverSocket.accept()

print "Connection form: ", clientAddress

while True:

sentence = connectionSocket.recv(1024)

if sentence == '.':

break

print len(sentence)

# connectionSocket.send(capitalizedSentence)

connectionSocket.close()

**TCP server thread**

from socket import \*

import thread

def handler(connectionSocket):

while True:

sentence = connectionSocket.recv(1024)

if sentence == '.':

break

capitalizedSentence = sentence.upper()

connectionSocket.send(capitalizedSentence)

connectionSocket.close()

serverPort = 12000

serverSocket = socket(AF\_INET, SOCK\_STREAM)

serverSocket.setsockopt(SOL\_SOCKET, SO\_REUSEADDR, 1)

serverSocket.bind(('', serverPort))

serverSocket.listen(1)

while True:

print 'The server is ready to receive'

newSocket, addr = serverSocket.accept()

thread.start\_new\_thread(handler, (newSocket,))