Università degli Studi di Trieste

Corso di ingegneria industriale

Esercitazioni di Fondamenti di Informatica

Giacomo Strangolino

mailto/chat: delleceste@gmail.com

Materiale didattico su:

http://www.giacomos.it

(http://www.giacomos.it/didattica/units/2013/)

Lezione 8 (20/11/2013)
Reti e programmazione multi thread

- Slide “Internetworking in ambito TCP/IP”

- Slide “Il livello di trasporto, TCP, UDP”

Programmazione rete: *socket* in Python – **Lato server (I)**

- Slide **“Il livello di trasporto, TCP, UDP”, n. 13-15.**

Programmazione rete: *socket* in Python – **Lato server (II)**

```
import socket
# crea un punto di comunicazione
tcpsock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Quando la connessione viene chiusa, il socket viene lasciato in uno stato
# di TIME_WAIT e non puo` essere utilizzato di nuovo immediatamente.
# Il messaggio di errore e`: “Address already in use”
# Si puo` dire al server di riutilizzarlo pero`:
#
tcpsock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
# associa un indirizzo locale a un socket
tcpsock.bind(self.address_tuple)
# si rende disponibile ad accettare connessioni
tcpsock.listen(0)
```

Programmazione rete: *socket* in Python – **Lato server (III)**

while True:

```
    # accetto una nuova connessione
    # clientsock: socket utilizzabile per ricevere e mandare dati
    # con quel particolare client
    # cliaddr:    l'indirizzo del client associato al socket

    (clientsock, cliaddr) = tcpsock.accept()

    # tipicamente, una volta che la accept accetta
    # una nuova connessione da un
    # client remoto, va creato un thread per gestire
    # quella particolare connessione
```

tcpsock.close() # alla fine

Programmazione rete: *socket* in Python – **Lato client**

```
# Creo un punto di comunicazione
# socket.AF_INET e socket.SOCK_STREAM sono
# numeri interi definiti nella classe socket.
socket = socket.socket(socket.AF_INET,
                       socket.SOCK_STREAM)

# Tenta di stabilire una connessione
socket.connect("140.105.5.83", 20022)

# … lettura dal socket e scrittura sul socket per scambiare
# … dati con il server... vedi prossime slide


tcpsock.close()  # alla fine
```

socket in Python – **scrittura**

Una volta che i *socket* sono stati correttamente creati (dopo che la *accept()* sul server e la *connect()* sul client sono andate a buon fine)
#
invio di dati sottoforma di *byte*
indifferentemente se e` il server a mandarli al
client o viceversa
la funzione bytes trasforma un oggetto
string in byte
data = bytes(“messaggio di testo in bytes”)

mando i byte attraverso il socket
socket.sendall(data)

socket in Python – **lettura (I)**

```
# Una volta che i socket sono stati correttamente
creati (dopo che la accept() sul server e la
connect() sul client sono andate a buon fine)
#
# lettura di dati sottoforma di byte
# indifferentemente se e` il server a riceverli dal
# client o viceversa
# la recv() si blocca (come la input() per
# la lettura da tastiera) finché non riceve dati
# ricevo oggetti di typo byte
data = socket.recv(2048)
# da byte converto a stringa, decodifica utf-8
 = data.decode('utf-8')
```

socket in Python – **lettura (II)**

```
while len(data):
    # utilizzo la stringa ricevuta
    fai_qualcosa(data_str)

    # poi ascolto sul socket finché
    # non arrivano nuovi dati
    data = self.socket.recv(2048)

# Tipicamente, per semplicità, il ciclo viene
# terminato quando l'utente preme i tasti
# Control e C insieme sulla tastiera, e il
# programma esce
```

socket in Python – **lettura (III)**

Nota: se il client invia una stringa di messaggio al server alla pressione di *invio* (quando la *input()* legge da tastiera), in generale ***non è detto che la recv() riceva tutto il blocco di byte assieme***.

E` opportuno quindi, per ogni blocco di dati ricevuto, cercare il carattere di “a capo” ('/n') e utilizzarlo come carattere di terminazione di un messaggio

socket in Python – **lettura (IV)**

```
message = ""
[...]
while len(data):
        data_str = data.decode('utf-8')
        endOfDataIdx = data_str.find('\n')
        while endOfDataIdx > -1:
            # trovato il terminatore: copio la fine del messaggio
            message += data_str[:endOfDataIdx]

            # gestisco il messaggio ricevuto
            fai_qualcosa(message)

            # ripulisco message
            message = ""
            # cerco altri '\n' se presenti (cio` accade se in un
            # buffer di dati arrivano piu` messaggi insieme)
            data_str = data_str[endOfDataIdx + 1:]

            endOfDataIdx = data_str.find('\n')
```

Thread

• E` una suddivisione di un processo in due o più filoni o *sottoprocessi*, che vengono eseguiti *concorrentemente*

• è contenuto all'interno di un processo;

• diversi *thread* contenuti nello stesso processo **condividono** alcune **risorse** (*ad esempio i dati*)

→ Più *thread* possono accedere ad una stessa variabile e modificarne il contenuto o valore.

Thread (II)

• può accadere che un *thread* modifichi il valore di una variabile, mentre un altro *thread* necessita del vecchio valore memorizzato in essa.

• Si ricorre all'uso di *tecniche di sincronizzazione* come la mutua esclusione per risolvere il problema (*lock, mutex (semafori)*)

• idealmente, un *thread* dovrebbe eseguire del codice quanto più possibile *indipendente* dal resto del programma

→ per il progetto chat, si trascuri l'aspetto della sincronizzazione

Thread in python

- Serve il modulo *threading*
  → **import threading**

- Si eredita dalla classe Thread del modulo threading
  → **class ReadingThread(threading.Thread):**

```
# nel caso chat vorro` condividere il
# socket con il thread
def __init__(self, socket, …):
  threading.Thread.__init__(self)
  self.socket = socket
  # ...
```

Thread in python (II)

- Si deve implementare il metodo **run()** della classe threading.Thread

→ **class ReadingThread(threading.Thread):**

```
def __init__(....)
    ....

# read from socket and process data
def run(self):
  # leggo al più 2048 byte alla volta
  data = self.socket.recv(2048)
  while len(data):
    data_str = data.decode('utf-8')
    ...
```

Thread in python (III)

- Creazione di un *thread* e sua esecuzione all'interno del thread principale:

```
# istanzio un thread di tipo ReadingThread
self.readingThread = ReadingThread(self.socket, …)
# e lo eseguo (il metodo start() esegue il metodo run()
# del thread)
self.readingThread.start()
```

Esempi e documentazione python

- Esempio *threads.py* nella cartella lezione 8
- Esempio *echo_cliserv.py* nella stessa cartella
- http://docs.python.org/3/library/threading.html
- http://docs.python.org/3/library/socket.html

Curiosità

- *print()* a colori (funziona anche su windows?)
- print(“\033[1;32mhello!\033[0m”)

  → “\033[1;32m” carattere *escape* per stampare in verde (*escape*, analogamente a “\n”, “\t” (tab))

  → “\033[0m” per resettare al colore di default
- “\033[1;31m” rosso
- “\033[1;33m” giallo
- “\033[1;34m” blu
- “\033[1;35m” viola
- “\033[1;36m” azzurro

Lezione 9
Homework

Si progetti un gioco per la battaglia navale.

- Vi saranno N navi, rappresentate ciascuna dall'oggetto *Nave*, con una posizione della prua, una dimensione e un orientamento (orizzontale o verticale) sulla scacchiera.

  La classe Nave dovrà avere dei metodi che dicano se è *colpita(x, y)* o addirittura *affondata()*

- Un oggetto *Scacchiera* conterrà le N navi in una lista e dovrà fornire il metodo *colpisci(x, y)* che chiede ad ogni nave se *(x,y)* è una posizione che la danneggia o la affonda nel caso estremo.

  La *Scacchiera* avrà altresì il metodo *disegna()* che ad ogni mossa eseguita dal giocatore deve disegnare la battaglia navale sotto forma di matrice contenente opportuni simboli adatti a descrivere la situazione corrente...

Lezione 9
Homework

Suggerimenti

- Si usi la funzione *randint(a,b)* del modulo random per generare un numero casuale tra 0 e N (ne serviranno 3, uno per la posizione x della prua, uno per la y e uno (0 oppure 1) per l'orientazione (orizzontale o verticale) di ogni nave.
- Se durante la generazione delle navi due navi si scontrano, bisogna rigenerare l'ultima nave finché non ci sono più intersezioni...
- La funzione che usa la classe Scacchiera dovrà eseguire un ciclo infinito finché o tutte le navi sono affondate o l'utente inserisce una stringa particolare, tipo “esci”, per terminare anticipatamente il gioco.