Programmazione di reti

Traccia 2
Architettura client-server UDP per trasferimento file

Domenico Francesco Giacobbi

domenico.giacobbi@studio.unibo.it

988158

Indice:

1 Scelte progettuali:

1.1 Scelta della dimensione del buffer

I socket si avvalgono di specifici buffer per effettuare lo scambio di dati.
La scelta della loro dimensione ricopre particolare importanza e va effettuata prendendo in considerazione la velocità di riempimento e svuotamento del buffer, in modo tale da migliorare le performance della comunicazione.
Nell'applicazione viene utilizzato come protocollo di comunicazione UDP, di conseguenza ho impostato la dimensione dei pacchetti inviati pari a quella del buffer di comunicazione, in modo tale da evitare che i pacchetti vengano silenziosamente scartati.
I socket DGRAM risentono particolarmente dell'eccedenza di bytes.

1.2 Gestione di più client

Il server gestisce simultaneamente le richieste effettuate da piu' client.
Quando il server riceve una determinata richiesta da parte di un client, gli viene assegnato un numero di porta dedicato e viene istanziato un thread che esegue in parallelo le operazioni richieste.

In particolare per ogni client viene creato un apposito socket DGRAM(che utilizza il protocollo udp), a cui viene associato l'indirizzo ip del server e il numero di porta precedentemente generato.
Le operazioni richieste dal client ed eseguite nei thread utilizzano per la trasmissione di dati tra il server e il client il nuovo socket istanziato.
Il nuovo indirizzo a cui è associato il socket viene inviato inizialmente dal server al relativo client.

1.3 Mutua esclusione

L'applicazione, come spiegato precedentemente, è un'applicazione multithread.
I thread per effettuare correttamente le operazioni a loro sottoposte non devono mai utilizzare contemporaneamente le stesse risorse, le operazioni devono essere eseguite in mutua esclusione.

Ciò è possibile in python utilizzando la classe Lock, la quale possiede al suo interno i metodi acquire e release.
Il metodo acquire permette di limitare l'esecuzione di una porzione di codice ad un unico thread, gli altri thread entrano in attesa che le risorse siano state liberate prima di potervi accedere.
Il metodo release rilascia le risorse bloccate mediante il metodo acquire, rilasciandole ad un altro thread che ne fa richiesta.
All'interno del server ho utilizzato tale tecnica per evitare che più client effettuando la stessa richiesta utilizzino le stesse risorse contemporaneamente.

1.4 Connessioni tra client e server

Inizialmente ogni client invia al server un messaggio per notificargli il suo indirizzo.
Il server assegna le operazioni richieste dal client ad un apposito thread, nel quale è presente un socket a cui è assegnato l'indirizzo generato secondo le modalità spiegate precedentemente.

All'interno del thread viene inviato un messaggio di conferma al client, mediante il quale esso assume l'indirizzo associato al nuovo socket.
Da quel momento in poi il server e il client possono iniziare a scambiarsi pacchetti in base alle operazioni da effettuare.
In particolare viene gestito il download della lista dei file presenti nel server, il download di un file presente nel server e l'upload di un file dal client al server.

1.5 Connessioni tra client e server

Inizialmente nel client vengono richiesti l'indirizzo ip e il numero di porta del server a cui deve connettersi.
Per verificare che l'indirizzo sia corretto viene impostato un timeout di pochi secondi e viene inviato un messaggio di notifica al server.
Se il client non riceve una risposta dal server entro il tempo impostato l'indirizzo viene considerato errato e viene nuovamente richiesto all'utente

1.6 Interfaccia grafica

Per rendere più user friendly l'applicazione è stata realizzata un'interfaccia grafica mediante tkinter
Essa permette di interfacciarsi all'applicativo in modo più semplice rispetto alla linea di comando, permettendo un'esecuzione agevolata anche da parte di persone meno avvezze all'informatica.

2 Strutture dati

Classi utilizzate:

Per implementare il client dell'applicazione ho utilizzato le seguenti classi:

- Client
  La quale assembla la logica e l'interfaccia grafica del client e permette la sua esecuzione, mediante i suoi metodi e i suoi attributi
- ViewClient
  Implementa l'interfaccia grafica del client, mediante i suoi metodi ed i suoi attributi
- ModelClient
  Implementa la logica del client, mediante i suoi metodi ed i suoi attributi.
  Per logica si intendono le operazioni vere e proprie eseguite dal client

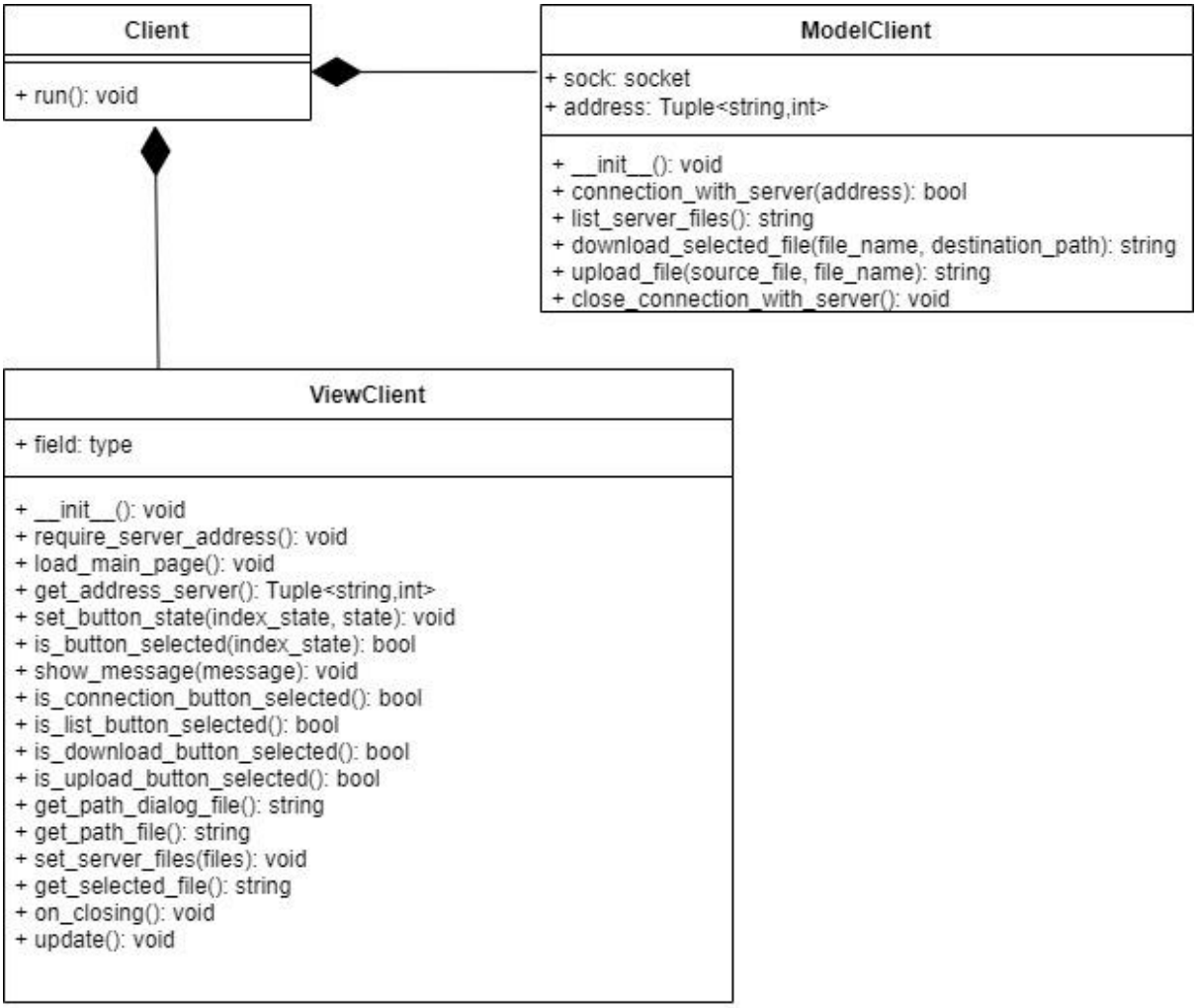
Per implementare il server dell'applicazione ho utilizzato le seguenti classi:

- Server
  La quale assembla la logica e l'interfaccia grafica del client e permette la sua esecuzione, mediante i suoi metodi e i suoi attributi
- ViewServer
  Implementa l'interfaccia grafica del server, mediante i suoi metodi ed i suoi attributi
- ModelServer
  Implementa la logica del server, mediante i suoi metodi ed i suoi attributi.
  Per logica si intendono le operazioni vere e proprie eseguite dal server

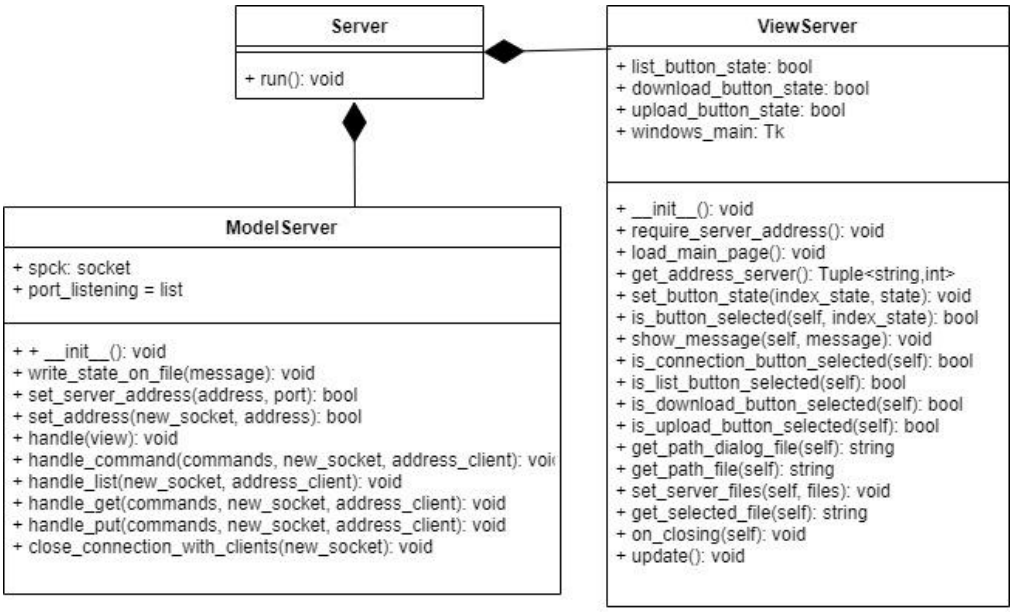
2.1 Schema UML

Nei seguenti schemi sono presenti solo gli attributi principali

client:

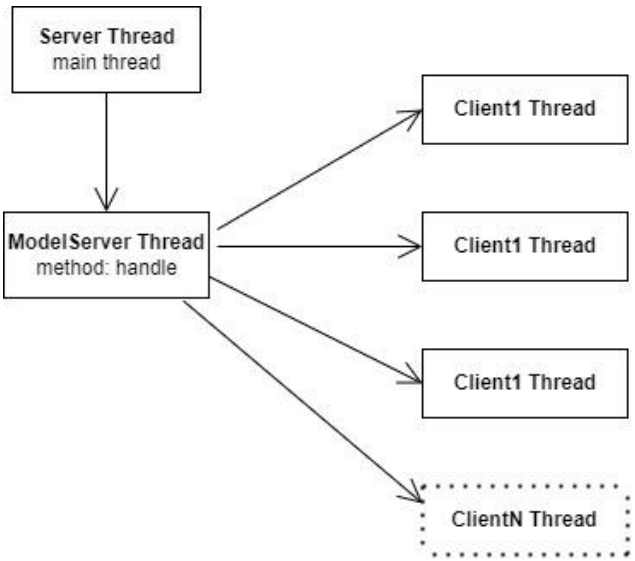


server:



3 Thread:

Il thread principale esegue la classe model server in un thread esterno, in quanto la view utilizza il metodo mainloop e di conseguenza i suoi metodi non possono essere richiamati dall'esterno.
Per ogni client ,nella classe ModelServer, il metodo handle istanzia un thread a cui vengono delegate le richieste effettuate.
Di conseguenza è in esecuzione un thread per ogni client attivo.



4 .Conclusioni:

L’applicazione permette lo scambio di dati tra client e server correttamente senza particolari problemi.
Tuttavia essi sono eseguiti in locale sulla stessa macchina, inserendoli su due macchine differenti e in una rete piu’ complessa l’applicazione inizierebbe a dare problemi.
Infatti per lo scambio di pacchetti viene implementato mediante il protocollo UDP, il quale è un protocollo connection-less e che quindi non garantisce l’arrivo ordinato dei dati a destinazione.
Per un applicativo di questo genere sarebbe piu’ corretto utilizzare il protocollo TCP