RETI INFORMATICHE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

SPECIFICHE DI PROGETTO A.A. 2017/2018

Il progetto consiste nello sviluppo di un'applicazione client/server scritta in C. Nello specifico, l'applicazione da sviluppare è un sistema di "**messaggistica"** che permette lo scambio di messaggi tra diversi utenti (i clients). Il sistema dovrà implementare due modalità di scambio dei messaggi: scambio di messaggi istantaneo e scambio di messaggi off-line. Lo scambio di messaggi *istantaneo* avviene in maniera diretta tra gli utenti attraverso interazioni peer2peer via socket UDP, questa modalità può essere scelta solo quando gli utenti coinvolti nella conversazione sono entrambi online. Lo scambio di messaggi *off-line*, invece, avviene in maniera indiretta attraverso la comunicazione client/server via socket TCP e sarà impiegata quando l'utente destinatario del messaggio non è online. In questa modalità il server dovrà farsi carico di memorizzare il messaggio e trasmetterlo al momento della connessione dell'utente destinatario.
Il server avrà il compito di memorizzare i dati degli utenti registrati e, tra questi, quali sono online al momento. Per gli utenti online il server dovrà memorizzare l'indirizzo IP e la porta UDP sulla quale il client è in ascolto in modo da fornire queste informazioni ad altri clients che vogliono inviare un messaggio in modalità istantanea. Lo scambio di informazioni tra client e server dovrà avvenire tramite socket TCP. La porta e l'indirizzo IP del server sarà fornita a riga di comando al momento del lancio del client.

Per sviluppare l'applicazione devono essere realizzati due programmi, **msg_server** per il lato server e **msg_client** per il lato client.

PROGRAMMA CLIENT

Il client deve essere avviato con la seguente sintassi:

```
./msg_client <IP locale> <porta locale> <IP server> <porta server>
```

dove:

- **<IP locale>** è l'indirizzo IP locale del client;
- **<porta locale>** è la porta locale su cui il client potrà ricevere messaggi istantanei (Porta UDP);
- **<IP server>** è l'indirizzo dell'host su cui è in esecuzione il server;
- **<porta server>** è la porta su cui il server è in ascolto (Porta TCP).

All’avvio, i comandi disponibili per l’utente devono essere:

- **!help**
- **!who**
- **!quit**
- **!register username**
- **!deregister**
- **!send username**

Il client deve stampare tutti gli eventuali errori che si possono verificare durante l’esecuzione.
Un *esempio* di esecuzione è il seguente:

```
$ ./msg_client 5555 127.0.0.1 1234

Connessione al server 127.0.0.1 (porta 1234) effettuata con successo
Ricezione messaggi istantanei su porta 5555

Sono disponibili i seguenti comandi:
!help --> mostra l'elenco dei comandi disponibili
!register username --> registra il client presso il server
!deregister --> de-registra il client presso il server
!who --> mostra l'elenco degli utenti disponibili
!send username --> invia un messaggio ad un altro utente
!quit --> disconnette il client dal server ed esce
```

Nel caso in cui il nome utente sia già registrato presso il server, la connessione dovrà essere gestita come una riconnessione e il server dovrà inviare eventuali messaggi memorizzati offline[1]. La disconnessione attraverso il comando quit non comporta una de-registrazione ma solamente la messa in stato offline del client.

Comandi base

!help: mostra l’elenco dei comandi disponibili.
Esempio di esecuzione:

```
Sono disponibili i seguenti comandi:
!help --> mostra l'elenco dei comandi disponibili
```

[1] Si tralascino le implicanze in termini di sicurezza dovute alla mancanza di una qualsiasi forma di autenticazione (ad esempio basata su password).

```
!register username --> registra il client presso il server
!deregister --> de-registra il client
!who --> mostra l'elenco degli utenti disponibili
!send username --> invia un messaggio ad un altro utente
!quit --> disconnette il client dal server ed esce
```

!register username: il client si registra con il nome presso il server. Nel caso in cui un utente al momento offline sia già registrato la registrazione deve essere interpretata dal server come una riconnessione di un client. Nel caso in cui invece ci sia un'utente online con lo stesso nome, deve essere inviato un messaggio di errore. Subito dopo la registrazione vengono inviati eventuali messaggi memorizzati sul server per lo scambio in modalità offline.
Esempio di esecuzione:

```
> !register Pippo
Registrazione avvenuta con successo
Pluto (msg offline)>
Ciao!
Caio (msg offline)>
come stai?
Pippo>
```

!deregister: il client si de-registra usando il nome corrente.
Esempio di esecuzione:

```
Pippo> !deregister
Deregistrazione avvenuta con successo
>
```

!who: mostra l'elenco dei client registrati e il loro stato, online o offline.
Il server mantiene una lista dei client registrati e dei client connessi in ogni momento. La lista dei client connessi contiene l'indirizzo IP e la porta di ascolto UDP con cui i client si sono registrati al momento della registrazione. Questi dati saranno forniti al client ogni volta che questo lo richieda per lo scambio di messaggi di tipo istantaneo.
Esempio di esecuzione:

```
Pippo> !who
Client registrati:
   client1(online)
   client2(offline)
```

```
client3(offline)
client4(offline)
Pippo>
```

!send username: il client invia un messaggio all'utente specificato. Se l'utente è online il messaggio verrà inviato in modalità istantanea, altrimenti il messaggio viene inviato al server per l'invio in modalità offline.

Gli errori da gestire sono:

- username inesistente
- errori a livello protocollare

Più in dettaglio il client farà richiesta al server (sempre tramite TCP) per sapere se esiste l'utente **username**. Se l'utente è online il server risponderà con l'indirizzo IP e porta di ascolto UDP del client **username**. Se l'utente invece è offline il server risponderà con un messaggio apposito dando la possibilità al client di inviare il messaggio in modalità offline attraverso il server. Il messaggio dovrà essere scritto subito dopo il comando, terminato dal carattere '.' digitato su una linea singola.

Esempio di esecuzione:

```
Pippo> !send Pluto

Ciao Come stai?
Io tutto bene.
.
Messaggio istantaneo inviato.
Pippo>
```

Sul client destinatario:

```
Pippo (msg istantaneo)>
Ciao Come stai?
Io tutto bene.
Pluto>
```

Possibile segnalazione di errore:

```
Pippo> !send username

Impossibile connettersi a username: utente inesistente.
```

!quit: il client chiude il socket TCP, il socket UDP ed esce. Il server stampa un messaggio che documenta la disconnessione del client. Il server, dovrà gestire in maniera appropriata la disconnessione di un cliente,

cambiando lo stato in offline.

```
> !quit


Client disconnesso
```

LATO SERVER

Il server **msg_server** si occupa di gestire le richieste provenienti dai client e di memorizzare i messaggi offline e trasmetterli alla prima riconnessione del client destinatario. Il server dovrà gestire client multipli connessi allo stesso tempo.
La sintassi del comando è la seguente:

```
./msg_server <porta>
```

Dove **<porta>** è la porta su cui il server è in ascolto. Il server dovrà essere raggiungibile da tutti gli indirizzi IP della macchina su cui è in esecuzione.
Una volta eseguito, **msg_server** deve stampare a video delle informazioni descrittive sullo stato del server (creazione del socket di ascolto, connessioni accettate, operazioni richieste dai client ecc.).

REQUISITI

- Client e server si scambiano dati tramite socket TCP, i client si scambiano dati tra loro tramite UDP. Prima che inizi ogni scambio è necessario che il ricevente sappia quanti byte deve leggere dal socket. **Non è ammesso che vengano inviati su socket numeri arbitrari di byte**.
- Per memorizzare i messaggi offline e i dati degli utenti connessi si utilizzi sul server la struttura dati che si preferisce. In generale si lascia libertà di implementazione per ciò che non è esplicitamente specificato. Le strutture dati possono essere allocate sfruttando la memoria dinamica, oppure staticamente fissando un numero massimo. Nel secondo caso si dovranno gestire eventuali errori.
- Si utilizzino gli autotools (comando make) per la compilazione del progetto.

CONSEGNA DEL PROGETTO

Il progetto dovrà essere consegnato almeno 24h prima dell'esame attraverso il sistema elearn.ing.unipi.it sulla pagina apposita del corso (http://elearn.ing.unipi.it/course/view.php?id=1079) usando le proprie credenziali di ateneo. Per ogni appello verrà creato una nuova sezione apposita per le consegne.

VALUTAZIONE DEL PROGETTO

Il progetto viene valutato durante lo svolgimento dell'esame. Il tutto verrà fatto girare su una macchina con

sistema operativo **Debian 8**. Si consiglia caldamente di testare il sorgente su una macchina Debian 8 prima di venire all'esame.

La valutazione prevede le seguenti fasi:

1. **Compilazione dei sorgenti**. Il client e il server vengono compilati attivando l'opzione **`-Wall`** che abilita la segnalazione di tutti i warning, e **non dovranno presentare warning o errori**. Si consiglia vivamente di usare tale opzione anche durante lo sviluppo del progetto, **interpretando i messaggi forniti dal compilatore.**
2. **Esecuzione dell'applicazione**. Il client e il server vengono eseguiti simulando una tipica sessione di utilizzo. In questa fase si verifica **il corretto funzionamento dell'applicazione e il rispetto delle specifiche fornite.**
3. **Esame del codice sorgente**. Il codice sorgente di client e server viene esaminato per controllarne l'implementazione. **Può essere richiesto di apportare modifiche al codice sul momento.**