SISTEMI DI ELABORAZIONE
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA ELETTRONICA

SPECIFICHE DI PROGETTO A.A. 2019/2020, v.1.0

Il progetto deve essere realizzato singolarmente (non è possibile realizzarlo in gruppo).

Il progetto consiste nello sviluppo di un'applicazione client/server. Client e server devono comunicare tramite socket TCP. Il server deve essere concorrente e la concorrenza deve essere implementata con i thread POSIX. Il thread principale deve rimanere perennemente in attesa di nuove connessioni in arrivo. Ogni connessione accettata deve essere smistata verso un membro di un pool di thread preallocati, che hanno il compito di gestire le richieste.

L'applicazione da realizzare, che chiameremo *Robin*, è un servizio di microblogging e social networking. Ogni utente che vuole utilizzare Robin deve per prima cosa eseguire una procedura di registrazione in cui specifica il suo nome utente e una password. Una volta registrato, l'utente può accedere a Robin mediante una fase di login basata sulle credenziali specificate durante la fase di registrazione.
L'utente a questo punto può

- inviare messaggi composti da al più 280 caratteri (detti *cip*); un cip può contenere delle parole che cominciano per # e che denotano gli argomenti del messaggio;
- diventare seguace di altri utenti;
- visualizzare la propria *home*.

Quando un utente visualizza la sua home vengono mostrati:

- il numero dei suoi seguaci;
- i messaggi inviati nell'ultima ora dalle persone che lui segue, visualizzati in ordine cronologico inverso;
- la lista dei dieci argomenti più caldi, intesi come quelli che sono stati menzionati il maggior numero di volte all'interno dei cip di tutti gli utenti nelle ultime 24 ore.

L'intero sistema è, per semplicità, non persistente con l'esclusione delle informazioni relative alle registrazioni degli utenti.

LATO CLIENT

Il client viene avviato con la seguente sintassi:

```
./robin_client <host remoto> <porta>
```

dove

- `<host remoto>` è l'indirizzo dell'host su cui è in esecuzione il server;
- `<porta>` è la porta su cui il server è in ascolto.

I comandi per l'utente sono:

- `help`
- `register`
- `login`
- `follow`
- `cip`
- `home`
- `quit`

Significato ed esempi di utilizzo dei comandi:

1. `help`: mostra l'elenco di comandi disponibili. Esempio di esecuzione:

```
>help
Comandi disponibili:
help                    --> mostra l'elenco dei comandi disponibili
register                --> registrazione al sistema
login             --> entra nel sistema
follow altro_utente     --> diventa seguace di altro_utente
cip                     --> nuovo cip
home                    --> visualizza la mia home page
quit                    --> esce dal sistema
>
```

2. `register`: registra un nuovo utente nel sistema. La registrazione può fallire nel caso in cui un utente con lo stesso nome si sia registrato in precedenza. Esempio di esecuzione:

```
>register
Inserisci il tuo indirizzo di posta elettronica:
mario.rossi@abcd.com
Scegli la tua password:
pippo
Registrazione fallita, utente già presente.
>
```

   Altro esempio:

```
>register
Inserisci il tuo indirizzo di posta elettronica:
marco.bianchi@efghi.com
Scegli la tua password:
pluto
Registrazione eseguita.
```

```
>
```

3. `login`: autenticazione di un utente precedentemente registrato. Esempio di esecuzione:
```
>login
Inserisci il tuo nome utente (indirizzo di posta elettronica):
marco.bianchi@eghi.com
Inserisci la password:
pluto
Benvenuto!
>
```

   La procedura di autenticazione fallisce nel caso in cui l'utente non sia registrato, se la password non è corretta o se l'utente ha già eseguito login. Esempio:
```
>login
Inserisci il tuo nome utente (indirizzo di posta elettronica):
marco.bianchi@eghi.com
Inserisci la password:
asdasdasd
Password non corretta!
>
```

4. `follow` *`nomeutente`*: l'utente diventa seguace di quello specificato come argomento. Esempio di esecuzione:
```
>follow mario.verdi@xyz.com
Adesso segui mario.verdi@xyz.
>
```

   Per poter eseguire il comando *follow* l'utente deve aver precedentemente eseguito con successo la procedura di *login*.

5. `cip`: invia un messaggio di al più 280 caratteri. Per poter eseguire il comando *cip* l'utente deve aver prima eseguito la procedura di *login*.
   Esempio:
```
>cip
Oggi vado a fare una #passeggiata in #montagna
```

   All'interno di un cip è possibile inserire delle parole che cominciano per # e che indicano gli argomenti del cip.

6. `home`: mostra a video la home dell'utente. Per poter eseguire il comando *home* l'utente deve aver già eseguito la procedura di *login*. Esempio di esecuzione:
```
>home
-----------------------------
Hai 8 seguaci
- - - - - - - - - - - - - - -
```

```
Messaggi:
12:31, marco@pippo.com: Ho preparato le #lasagne
12:28, franca@pluto.it: Che bello passeggiare in riva al #mare
10:04, arturo@xyz.edu: Non vedo l'ora di andare al #concerto
                       dei #cuginidimontagna
- - - - - - - - - - - - - - -
Argomenti caldi:
1 lasagne (33)
2 unix (21)
3 mare (8)
...
10 concerto (2)
------------------------------
>
```

Nella classifica dei dieci argomenti caldi il numero tra parentesi indica il numero di volte che è stato incluso nei cip postati nelle ultime 24 ore.

7. `quit`: l'utente esce da Robin e il client si disconnette. Esempio:
   ```
   >quit
   Sei uscito dal sistema.
   ```

LATO SERVER

Il server `robin_server` è concorrente ed utilizza thread ausiliari per gestire le richieste che provengono dai client. Il thread main, prima di mettersi in attesa di connessioni, prealloca un pool di thread ausiliari. Il thread main assegna ad un thread ausiliario (libero) del pool ogni nuova connessione che riceve. Il numero di thread del pool è specificato a tempo di compilazione (è una costante e non varia durante l'esecuzione del programma).

La sintassi del comando è la seguente:

```
./robin_server <host> <porta>
```

dove:

- `<host>` è l'indirizzo su cui il server viene eseguito;
- `<porta>` è la porta su cui il server è in ascolto;

Possiamo schematizzare lo schema di funzionamento del server nel seguente modo:

1. il thread main crea NUM_THREAD thread gestori;
2. il thread main si mette in attesa di connessioni in arrivo;
3. quando riceve una connessione, il thread main:

    a. controlla il numero di thread occupati (quelli che stanno attualmente eseguendo una richiesta);
    b. se tutti i thread del pool sono occupati si blocca in attesa che uno diventi libero;
    c. se c'è almeno un thread libero ne sceglie uno (senza nessuna politica particolare) e lo attiva;
4. il thread gestore (all'infinito):
    a. si blocca finché non gli viene assegnata una richiesta;
    b. riceve il comando da un client;
    c. esegue la richiesta del client;
    d. se il thread ha ricevuto il comando `quit`, il thread torna libero e a disposizione per servire un altro client. Altrimenti, il thread si blocca in attesa di un nuovo comando dallo stesso client.

Ne consegue che ogni thread gestore è associato ad uno e ad un unico client per tutto il tempo che quest'ultimo è connesso al server.
Una volta mandato in esecuzione `robin_server` deve stampare a video delle informazioni descrittive sulle operazioni eseguite (creazione del socket di ascolto, creazione dei thread, connessioni accettate, operazioni richieste dai client, ecc.).

Un esempio di esecuzione del server è il seguente:

```
$ ./robin_server 127.0.0.1 1235
Indirizzo: 127.0.0.1 (Porta: 1235)
THREAD 0: pronto
THREAD 1: pronto
THREAD 2: pronto
THREAD 3: pronto
MAIN: connessione stabilita con il client 127.0.0.1:1235
MAIN: E' stato selezionato il thread 0
THREAD 0: ricezione comando register
MAIN: connessione stabilita con il client 127.0.0.1:1235
MAIN: E' stato selezionato il thread 1
THREAD 0: ricezione comando login
THREAD 1: ricezione comando cip
THREAD 1: ricezione comando home
...
```

Il server mantiene le informazioni degli utenti registrati in un file *users.txt*. Ogni riga del file contiene il nome di un utente e la password cifrata di quell'utente. Esempio:

```
marco.rossi@eghi.com ASD12edT7iO9VBK5
gino.verdi@ijk.com   QwC4GM7IsG76HhIw
```

AVVERTENZE E SUGGERIMENTI

Test.
Si testino le seguenti configurazioni:

- un client viene avviato quando alcuni thread gestori sono già occupati (ma ce n'è almeno uno libero);
- un client viene avviato quando non ci sono più thread gestori liberi.

Modalità di trasferimento dati tra client e server (e viceversa).
Client e server si scambiano dati tramite socket TCP. Prima che inizi ogni scambio è necessario che il ricevente sappia quanti byte deve leggere dal socket.

Ogni risorsa condivisa deve essere protetta da opportuni meccanismi semaforici.

Non utilizzare meccanismi di attesa attiva in nessuna parte del codice.
Esempio: Quando un client è in attesa che l'utente inserisca un comando, il thread corrispondente nel server si deve bloccare (l'operazione *recv()* è bloccante).

VALUTAZIONE DEL PROGETTO

Il progetto viene valutato durante lo svolgimento dell'esame. Il codice sarà compilato ed eseguito su una distribuzione Ubuntu Linux. Si consiglia di testare il programma su Ubuntu prima dell'esame. La valutazione prevede le seguenti fasi.

1. **Compilazione dei sorgenti**. Il client e il server devono essere compilati dallo studente in sede di esame utilizzando il comando gcc (non sono accettati Makefile) e attivando l'opzione *-Wall* che abilita la segnalazione di tutti i warning. Si consiglia di usare tale opzione anche durante lo sviluppo del progetto, interpretando i messaggi forniti dal compilatore e provvedendo ad eliminarli.
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.