SISTEMI DI ELABORAZIONE
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA ELETTRONICA

SPECIFICHE DI PROGETTO A.A. 2018/2019

Versione del 7/5/2019

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 del server 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 dei client.

Vogliamo realizzare un sistema per la gestione delle prenotazioni di un albergo. Le prenotazioni sono limitate all'anno solare 2020.

LATO CLIENT

Il client viene avviato con la seguente sintassi:

```
./hotel_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 gli utenti sono:

- `help`
- `register`
- `login`
- `reserve`
- `release`
- `view`
- `quit`

I comandi *reserve*, *release* e *view* possono essere eseguiti solo dopo aver eseguito il login. Significato 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 nel sistema
   login    --> autenticazione
   reserve  --> prenota una camera
   release  --> cancella una prenotazione
   view     --> visualizza le prenotazioni
   quit     --> esci 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 nome:
   mario.rossi
   Scegli la tua password:
   pippo
   Registrazione fallita, utente già presente
   >
   ```
   Altro esempio:
   ```
   >register
   Inserisci il tuo nome:
   luigi.bianchi.99
   Scegli la tua password:
   pippo
   Registrazione eseguita
   >
   ```

3. `login`: autenticazione di un utente precedentemente registrato. Esempio di esecuzione:
   ```
   >login
   Inserisci il tuo nome:
   luigi.bianchi.99
   Inserisci la password:
   pippo
   Benvenuto!
   >
   ```
   La procedura di autenticazione fallisce se l'utente non è registrato, se la password non è corretta o se l'utente ha già eseguito login. Esempio:
   ```
   >login
   Inserisci il tuo nome:
   luigi.bianchi.99
   Inserisci la password:
   ```

```
asdasdasd
Password non corretta!
>
```

4. `reserve` *`data`*: prenota una stanza, se disponibile, per la data specificata. Esempio di esecuzione:
   ```
   >reserve 31/01
   Prenotazione eseguita, ti è stata assegnata la stanza 17, il codice
   della tua prenotazione è A93FG
   >reserve 23/05
   Prenotazione non eseguita, l'albergo è al completo per la data scelta
   ```
   Il codice prenotazione è una stringa alfanumerica casuale di 5 caratteri generata dal server e associata alla prenotazione.

5. `release` *`data stanza codice`*: cancella la prenotazione relativa alla data e alla stanza indicate come argomenti. Deve essere anche specificato il codice relativo alla prenotazione da cancellare. Esempio:
   ```
   >release 31/01 17 ERTY4
    Cancellazione non eseguita, il codice non corrisponde
   >release 22/03 44 RT43W
    Prenotazione cancellata
   ```

6. `view`: mostra a video i dettagli delle prenotazioni eseguite dall'utente:
   ```
   >view
   31/01/2020, stanza 17, codice prenotazione A93FG
   18/06/2020, stanza 21, codice prenotazione 8GT4A
   >
   ```

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

LATO SERVER

Il server `hotel_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:

```
./hotel_server <host> <porta> <numero stanze>
```

dove:

- `<host>` è l'indirizzo su cui il server viene eseguito;
- `<porta>` è la porta su cui il server è in ascolto;
- `<numero stanze>` è il numero di stanze di cui è dotato l'albergo; le stanze sono inizialmente tutte libere.

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 `hotel_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:

```
$ ./hotel_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 reserve
THREAD 1: ricezione comando release
...
```

Il server mantiene l'insieme di 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:

```
mario.rossi           ASD12edT7iO9VBK5
luigi.bianchi.99      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.