

SISTEMI DI ELABORAZIONE

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA ELETTRONICA SPECIFICHE DI PROGETTO A.A. 2014/2015

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 main deve rimanere perennemente in attesa di nuove connessioni e le deve smistare ad un **pool di thread preallocati** che hanno il compito di gestire le richieste.

L'applicazione da sviluppare è il gioco della **BATTAGLIA NAVALE** seguendo il paradigma **client-server**. La mappa del gioco dovrà essere di 10x10. Ciascuna cella è rappresentata da una coppia di coordinate: una lettera e un numero (e.g., C4). Le navi che dovranno essere presenti nella mappa sono:

Numero Navi	Dimensione Nave
1	4
2	3
3	2
4	1

Per sviluppare l'applicazione devono essere realizzati due programmi: **nbattle_server** per il lato server e **nbattle_client** per il lato client. I client devono comunicare ogni operazione e mossa al server: è il server che gestisce lo scambio dei messaggi fra i client.

1. LATO CLIENT

Il client deve essere avviato con la seguente sintassi:

```
./nbattle_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 disponibili per l'utente devono essere:

- `!help`

- !who
- !create
- !join
- !quit
- !disconnect
- !show_enemy_map
- !show_my_map
- !hit *coordinates*

Il client deve stampare tutti gli eventuali errori che si possono verificare durante l'esecuzione. All'avvio della connessione il client **deve** inserire il suo username. Non è permesso che più giocatori connessi al server condividano lo stesso username. Il server **deve** registrare l'avvenuta connessione indicando quale thread è stato assegnato a quel client e ne gestirà le richieste.

Un esempio di esecuzione è il seguente:

```
$ ./nbattle_client 127.0.0.1 1234

Connessione al server 127.0.0.1 (porta 1234) effettuata con successo

Sono disponibili i seguenti comandi:
* !help --> mostra l'elenco dei comandi disponibili
* !who --> mostra l'elenco dei client connessi al server
* !create --> crea una nuova partita e attendi un avversario
* !join --> unisciti ad una partita e inizia a giocare
* !disconnect --> disconnetti il client dall'attuale partita
* !quit --> disconnetti il client dal server
* !show_enemy_map --> mostra la mappa dell'avversario
* !show_my_map --> mostra la tua mappa
* !hit coordinates --> colpisci le coordinate coordinates

Inserisci il tuo nome: client1
>
```

Un possibile messaggio in caso di errore di connessione è il seguente:

```
$ ./nbattle_client 127.0.0.1 1071

Impossibile connettersi a 127.0.0.1:1071
```

Implementazione dei comandi

a) !help: mostra l'elenco dei comandi disponibili.

Esempio di esecuzione:

```
Sono disponibili i seguenti comandi:  
* !help --> mostra l'elenco dei comandi disponibili  
* !who --> mostra l'elenco dei client connessi al server  
* !create --> crea una nuova partita e attendi un avversario  
* !join --> unisciti ad una partita e inizia a giocare  
* !disconnect --> disconnetti il client dall'attuale partita  
* !quit --> disconnetti il client dal server  
* !show_enemy_map --> mostra la mappa dell'avversario  
* !show_my_map --> mostra la tua mappa  
* !hit coordinates --> colpisci le coordinate coordinates
```

b) !who: mostra l'elenco dei client connessi.

Il server mantiene un'apposita struttura dati che contiene i client connessi in ogni momento. Per ciascun client, si memorizza lo username con cui il client si è registrato al momento della connessione e l'indice del thread che ne gestisce le richieste.

Esempio di esecuzione:

```
> !who  
Client connessi al server: client1 client3 client4
```

c) !create: il client comunica al server l'intenzione di dare inizio ad una nuova partita. Dopodiché il client resta in attesa che un avversario si unisca alla partita. Quando l'avversario è stato trovato, la partita inizia.**Esempio di esecuzione:**

```
> !create  
Nuova partita creata.  
In attesa di un avversario...  
client2 si è unito alla partita.  
La partita è iniziata.  
Inserisci le coordinate della 1a nave di dimensione 4: C4  
Inserisci l'orientamento della 1a nave di dimensione 4: VERTICALE  
Inserisci le coordinate della 1a nave di dimensione 3: F3  
Inserisci l'orientamento della 1a nave di dimensione 3: VERTICALE  
Inserisci le coordinate della 2a nave di dimensione 3: F8  
Inserisci l'orientamento della 2a nave di dimensione 3: ORIZZONTALE  
...  
Sta a te.
```

d) !join: il client comunica al server l'intenzione di sfidare un utente (che avrà preventivamente provveduto a creare una nuova partita tramite il comando !create). L'utente deve poter inserire lo username dell'avversario.

I possibili errori da gestire sono:

- lo username inserito è inesistente;
- l'avversario non è in ascolto (non ha eseguito il comando `!create`);
- l'avversario è già occupato in una partita;
- errori a livello protocollare.

Se l'operazione va a buon fine, la partita inizia automaticamente.

Esempio di esecuzione:

```
> !join
Inserire lo username dell'utente da sfidare: client1
La partita è iniziata.
Inserisci le coordinate della 1a nave di dimensione 4: G2
Inserisci l'orientamento della 1a nave di dimensione 4: ORIZZONTALE
Inserisci le coordinate della 1a nave di dimensione 3: A8
Inserisci l'orientamento della 1a nave di dimensione 3: VERTICALE
Inserisci le coordinate della 2a nave di dimensione 3: A1
Inserisci l'orientamento della 2a nave di dimensione 3: ORIZZONTALE
...

In attesa che client1 faccia la sua mossa...
client1 dice L10: ACQUA!

Sta a te.
```

Possibile segnalazione di errore:

```
> !join
Inserire lo username dell'utente da sfidare: client1
Impossibile connettersi a client1: utente inesistente.
```

```
> !join
Inserire lo username dell'utente da sfidare: client1
Impossibile connettersi a client1: l'utente è impegnato in altra partita.
```

- e) `!disconnect`: disconnette il client dall'attuale partita. Il comando è eseguibile soltanto durante una partita.

```
# !disconnect
Disconnessione avvenuta con successo: TI SEI ARRESO
```

Quando il client comunica al server la disconnessione, l'utente è di nuovo libero e può iniziare una nuova partita. Il server provvede a comunicare al client avversario che ha vinto

la partita per resa dello sfidante.

- f) `!quit`: il client chiude il socket con il server ed esce. Il server stampa un messaggio che documenta la disconnessione del client. Il server, inoltre, dovrà gestire in maniera appropriata la disconnessione del client.

```
> !quit
Client disconnesso correttamente
```

- g) `!show-my-map`: il client mostra lo stato attuale della propria mappa, visualizzando la posizione delle proprie navi e degli attacchi pervenuti fino a quel momento da parte dell'avversario. Il comando è eseguibile soltanto durante una partita.

Esempio di esecuzione:

```
# !show-my-map
10 - - - - X X - - X -
 9 - - X o - - - - -
 8 X - X - - o - X - -
 7 - - X - - o - - - X
 6 - - X - - H H H - X
 5 - - - - - o - - - -
 4 H - - o - - - - -
 3 o - X - - X X X - o
 2 o - X - - - - - H
 1 - - - - - - - - H
   A B C D E F G H I L
```

- X rappresenta una cella occupata da una propria nave che non è stata colpita dall'avversario
- H rappresenta una cella occupata da una propria nave che è stata colpita dall'avversario
- o rappresenta un tentativo dell'avversario andato a vuoto
- rappresenta una cella senza navi e sulla quale l'avversario non ha compiuto alcun tentativo

- h) `!show-enemy-map`: il client mostra lo stato attuale della mappa dell'avversario, visualizzando gli attacchi fatti fino a quel momento differenziando se sono state colpite navi oppure no. Il comando è eseguibile soltanto durante una partita.

Esempio di esecuzione:

```
# !show-enemy-map
10 - - - - -
9  - - - - - H -
8  - - - - - o - -
7  - o o - - - o - -
6  - - - - - - - -
5  - - - - o H H H o -
4  - - o - - - - - -
3  - - o - - o - - o -
2  - - - - - H H o
1  - - - - - - - -
    A B C D E F G H I L
```

- H rappresenta un tentativo che ha colpito una nave dell'avversario
- o rappresenta un tentativo andato a vuoto
- rappresenta una cella sulla quale non è stato compiuto alcun tentativo

- i) `#!hit coordinates`: colpisce la cella dell'avversario specificata dalle coordinate *coordinates*. Il comando è eseguibile soltanto durante una partita.

Esempio di esecuzione:

```
# !hit C4
client2 dice: COLPITO!
E' il turno di client2...
client2 dice F2: ACQUA!

Sta a te.
```

Fasi del gioco

1. Creazione della partita

Per iniziare una partita, è necessario che un utente abbia eseguito il comando `!create`, così da comunicare al server la sua disponibilità a giocare. Allo stesso tempo, il giocatore avversario deve eseguire il comando `!join`, specificando lo username dell'utente da sfidare. Così facendo, il server può comunicare al primo giocatore che un altro utente si è unito a lui per giocare. La partita può così iniziare.

Dal primo carattere della shell deve essere possibile capire se la partita è iniziata o meno:

```
> shell comandi      (accettati i comandi help, who, create, join, quit.
                      Se immesso altro viene restituito un errore)
# shell partita      (accettati i comandi help, show-my-map, show-enemy-map, hit,
                      disconnect. Se immesso altro viene restituito un errore)
```

2. Disposizione delle navi

Appena la partita inizia, i due giocatori scelgono la posizione delle proprie navi. In particolare, l'utente indica la cella iniziale e l'orientamento (verticale/orizzontale) per ciascuna nave. La nave si disporrà a partire dalla cella iniziale secondo l'orientamento indicato, verso celle con coordinate crescenti per un numero di celle pari alla dimensione della nave.

Esempio:

```
Inserisci le coordinate della 1a nave di dimensione 4: G2
Inserisci l'orientamento della 1a nave di dimensione 4: VERTICALE
...
```

La nave si disporrà su G2, G3, G4 e G5.

Il client deve effettuare un check per ogni posizionamento così da controllare se la nave appena impostata esce dalla mappa o si sovrappone ad un'altra nave. Se si verifica un errore il client avvisa l'utente ed aspetta di ricevere nuove coordinate. Le informazioni relative alle posizioni delle navi risiedono nel client e non vengono comunicate al server.

3. Effettuare un tentativo

Dopo aver disposto le proprie navi, i giocatori iniziano la sfida: a turno, cercano di indovinare la posizione delle navi avversarie. Mediante il comando `!hit`, il giocatore di turno può indicare la cella dell'avversario da colpire. Il client del giocatore che effettua il tentativo trasmette le coordinate al server, il quale le inoltra al client dell'avversario.

Quest'ultimo verifica se sulla cella indicata sia disposta una nave o meno e trasmette l'esito (COLPITO/ACQUA) al server, che lo inoltra all'altro giocatore. La verifica è eseguita in automatico via software senza l'intervento dell'utente.

[OPZIONALE] : è possibile implementare anche l'esito AFFONDATO.

A questo punto, cambia il turno di gioco. Il giocatore che aveva eseguito il comando `!hit` si mette in attesa della mossa dell'avversario.

Il client che ha creato la partita (tramite il comando `!create`) è il giocatore 1 ed inizia per primo. Il client che si è aggiunto come sfidante alla partita (tramite il comando `!join`) è il

giocatore 2 ed inizia per secondo. Durante il primo turno dunque aspetta il tentativo dell'altro giocatore.

4. `Attendere la mossa dell'avversario`

quando è il turno dell'avversario, il client si pone in attesa finché il giocatore sfidante non invia le coordinate del suo tentativo. Quando il server gli comunica le coordinate trasmesse dall'avversario, il client, come indicato precedentemente, verifica se sulla cella indicata sia disposta una nave o meno e manda l'esito (`COLPITO/ACQUA`) al server, che lo inoltra all'altro giocatore. A questo punto, cambia nuovamente il turno e il giocatore può fare un nuovo tentativo.

5. `Fine della partita`

Nel corso della partita si alternano ripetutamente le fasi 3 e 4. La partita termina quando un giocatore ha affondato tutte le navi dell'avversario, oppure abbandona la partita, oppure il suo avversario abbandona la partita.

2. LATO SERVER

Il programma `nbattle_server` si occupa di accettare nuove connessioni TCP, registrare nuovi utenti, gestire le richieste dei vari client per aprire nuove partite e gestire lo scambio di messaggi tra i client durante la partita. Il server `nbattle_server` è concorrente ed utilizza i thread per gestire le richieste. Il thread main, prima di mettersi in attesa di connessioni, prealloca un pool di thread gestori. Il thread main assegna ad un thread gestore (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:

```
./nbattle_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 delle connessioni in ingresso;
3. quando riceve una connessione in ingresso, il thread main:
 - a. controlla il numero di thread occupati (ovvero 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, oppure si blocca in attesa della mossa da parte dell'avversario del client servito.

Se ne ricava 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 eseguito, `nbattle_server` deve stampare a video delle informazioni descrittive dello stato del server (creazione del socket di ascolto, creazione dei thread, connessioni accettate, operazioni richieste dai client, ecc.).

Un esempio di esecuzione del server è il seguente:

```
$ ./nbattle_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: pippo si è connesso
MAIN: connessione stabilita con il client 127.0.0.1:1235
MAIN: E' stato selezionato il thread 1
THREAD 1: pluto si è connesso
THREAD 0: ricezione comando !create
THREAD 1: ricezione comando !join
THREAD 1: ricezione richiesta partita contro pippo
.....
```

3. 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. Non è ammesso che vengano inviati su socket numeri arbitrari di byte.

- **Ogni risorsa condivisa deve essere protetta da opportuni meccanismi semaforici**

- **Non sono accettati meccanismi di attesa attiva**

Quando un client è in attesa che l'utente inserisca un comando, il thread corrispondente nel server si blocca (l'operazione `recv()` è bloccante). Ugualmente, quando un client è in attesa della mossa da parte dell'avversario, il thread corrispondente nel server si blocca (e dovrà essere risvegliato al momento opportuno).

4. VALUTAZIONE DEL PROGETTO

Il progetto viene valutato durante lo svolgimento dell'esame. Il codice sarà compilato ed eseguito su sistema operativo Debian. Si consiglia di testare il sorgente su Debian prima dell'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*. Si consiglia 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.