Nome Cognome Matricola

Documentazione Trivia Quiz

Per il mio server ho scelto il protocollo TCP, perché la tipologia di applicazione dei quiz, simile a quella di messaggistica istantanea, non necessita urgenza nella consegna dei dati quanto piuttosto della trasmissione affidabile e ordinata di essi.

Mentre per quanto riguarda la tipologia del server, ho scelto l'I/O multiplexing e socket non bloccanti con i seguenti vantaggi:

- Rispetto ad un server iterativo, di poter gestire più connessioni simultaneamente in maniera efficiente senza dover attendere la risposta del singolo client.
- Rispetto ad un server concorrente oppure multi-threading, di non avere problemi di condivisione, sincronizzazione e concorrenza sulle strutture dati, in quanto il server è single thread.

Ma ciò a fronte di una maggiore complessità nella gestione dei client, in particolar modo la difficoltà è stata nel mantenere uno stato per ciascuno dei client, cosa che in un server concorrente sarebbe stata immediata.

Le domande vengono inviate in ordine insieme ad un array di booleani che rappresenta i topics giocabili dall'utente registrato.

La classifica è formata da più liste ordinate di punteggi, esse vengono serializzate e inviate in modalità testuale, a questo punto è semplice ritrasmettere la classifica a più client fino alla prossima modifica, le stesse serializzazioni sono usate nella stampa a video del server.

Ho cercato di mantenere una struttura stratificata nel codice: ogni dato più complesso di un comando viene convertito in un oggetto `Message`, che conserva il proprio stato di invio o ricezione. Ciò è particolarmente utile con l'IO multiplexing, poiché, se un socket non è pronto, il server può continuare a servire altri client e riprendere l'invio quando il socket torna disponibile

Istruzioni:

```
bin
├── client
├── data
│   ├── topics
│   │   ├── 01 - topic1.txt
│   │   ├── 02 - topic2.txt
│   │   └── 03 - topic3.txt
│   └── users
│       ├── user1.txt
│       ├── user2.txt
│       └── user3.txt
└── server
```

Ciascun topic può essere memorizzato in un file `txt` ciascuno sotto la cartella "`./data/topics/`", per l'ordinamento, è sufficiente numerare i file: il numero sarà ignorato una volta caricato.

L'inserimento delle domande può avvenire semplicemente alternando una domanda e la sua risposta, ed è possibile inserire una riga vuota tra ogni coppia e l'altra per migliorare la leggibilità.

Ciascun utente viene memorizzato in file separati sotto la cartella "`./data/users/`", dove ogni coppia di linee rappresenta il nome del topic e il suo punteggio.

Per la compilazione sarà sufficiente posizionarsi nella directory e digitare "`make`", e per quanto riguarda l'esecuzione è come da specifiche se ci si posiziona nella directory "`bin`" ovvero "`./server`" e "`./client 1234`".

I parametri del server sono nello header "`parameters.h`" presente nella directory "`src`".

Protocollo

```
typedef enum Command
{
    CMD_STOP = false, // Chiusura o errore
    CMD_NONE, // Niente o domande terminate
    CMD_OK, // Conferma positiva
    CMD_REGISTER, // Richiesta registrazione
    CMD_MESSAGE, // Messaggio, epilogo
    CMD_SCOREBOARD, // Classifica
    CMD_NOTVALID, // Nome client non valido
    CMD_EXISTING, // Nome client duplicato
    CMD_TOPICS, // Richiesta dati topics
    CMD_TOPICS_PLAYABLE, // Topic giocabili
    CMD_SELECT, // Selezione topic
    CMD_ANSWER, // Invio risposta
    CMD_NEXTQUESTION, // Prossima domanda
    CMD_CORRECT, // Risposta corretta
    CMD_WRONG, // Risposta errata
    CMD_FULL, // Server pieno
} Command;
```

La sincronizzazione nell'applicazione distribuita è fatta scambiando messaggi binari: un byte che rappresenta un enumerando "`Command`" viene scambiato per specificare ciascuna azione tra client e server.

Allo scopo di verificare che la logica tra server e client sia correttamente sincronizzata, viene anche inviato un byte di conferma, avendo però il difetto di raddoppiare il numero di turnaround time, nonostante la piccolezza dei messaggi scambiati.

La registrazione del client (dopo la effettiva `connect` ed `accept` da parte del server) avviene nel seguente modo:

```
Client → Server: CMD_REGISTER
Server → Client: CMD_OK
Client → Server: CMD_MESSAGE
Client → Server: 0x00 0x00 0x00 0x01
Client → Server: 0x00 0x00 0x00 0x05 "name" 0x00
Server → Client: CMD_OK
Server → Client: CMD_OK
```

Si può notare la doppia conferma da parte del server: la prima per la conferma avvenuta ricezione del messaggio e la seconda per la conferma registrazione (e nome accettato)

Nel caso il nome non fosse stato accettato il server avrebbe inviato un altro comando, e la procedura sarebbe stata ripetuta

Lo scambio di dati avviene tramite `MessageArray`, una struttura che contiene più messaggi. Si invia prima il numero di messaggi, poi per ciascuno la sua dimensione e il contenuto.

```
Client → Server: CMD_TOPICS
Server → Client: CMD_OK
Server → Client: CMD_MESSAGE 0x00 0x00 0x00 0x04 0x00 0x00 0x00 0x07 "storia" 0x00 0x00 0x00 0x00
0x08 "scienze" 0x00 0x00 0x00 0x00 0x0A "geografia" 0x00 0x00 0x00 0x00 0x03 0x01 0x01 0x01
Client → Server: CMD_OK
```

Si possono notare i gruppi di 4 byte che formano gli interi, trasmessi in big-endian.

Una volta registrato il client chiederà i topics tramite il comando `CMD_TOPICS` e verrà informato sia sui topics che su quelli giocabili (ultimi 3 byte del `MessageArray`).

```
Client → Server: CMD_OK CMD_SELECT
Server → Client: CMD_OK
Client → Server: CMD_MESSAGE 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x04 0x00 0x00 0x00 0x01
Server → Client: CMD_OK

Client → Server: CMD_NEXTQUESTION
Server → Client: CMD_OK
Server → Client: CMD_MESSAGE 0x00 0x00 0x00 CMD_NONE
Server → Client: 0x00 0x00 0x00 0x29 "Which planet is known as the Red Planet?" 0x00
```

```
Client → Server: CMD_ANSWER
Client → Server: CMD_MESSAGE
Client → Server: 0x00 0x00 0x00 0x01
Client → Server: 0x00 0x00 0x00 0x05
Client → Server: "mars" 0x00
Server → Client: CMD_OK
Server → Client: CMD_CORRECT
Client → Server: CMD_NEXTQUESTION
```

Dopo la trasmissione della domanda il client può chiedere la classifica inviando `CMD_SCOREBOARD` invece di `CMD_ANSWER` e riceverà il `MessageArray` con la serializzazione della classifica.