Progetto Reti Informatiche

A.A 24-25 (548611 – Giovanni Dipace)

Introduzione

Il progetto consiste nella realizzazione di un'applicazione distribuita che implementi un sistema di gaming distribuito. La connessione tra client e server è implementata esclusivamente con protocollo TCP, questo perché l'applicazione è suscettibile ad eventuali perdite di pacchetti, mentre la responsività, caratteristica del protocollo UDP, non è un requisito fondamentale per questo tipo di applicazione, in quanto il collo di bottiglia è l'interazione stessa con l'utente.

La modalità di scambio dei dati principale è quella text, questo perché la maggior parte dei messaggi inviati tra client e server sono stringhe di dimensioni variabili, scegliere quindi una dimensione prefissata del messaggio avrebbe portato o a sprecare memoria, nel caso si fossero usati buffer molto più grandi del necessario, o a costringere l'applicazione ad usare domande e risposte di una dimensione massima, nel qual caso si sarebbero usati buffer di dimensioni minime, ma questo avrebbe limitato di molto la varietà delle stesse. Alcune informazioni sono invece scambiate in modalità binary, questo perché sono di dimensioni fisse e sono utili al flusso del programma, come per la segnalazione della fine delle domande disponibili all'utente.

Server

Considerando i requisiti dell'applicazione un server concorrente è la scelta migliore, in quanto essa risulta maggiormente I/O bound: poiché gli utenti possono impiegare una quantità di tempo molto elevata per inserire i dati richiesti e le risposte, un server non concorrente li bloccherebbe tutti in attesa di un singolo, cosa che non è accettabile. Il server sfrutta i thread per fare in modo che i client vengano serviti contemporaneamente e in parallelo. La scelta di utilizzare i thread al posto dei processi con *fork()* deriva dalla facilità con cui è possibile condividere strutture dati tra i thread (cosa che risulta più difficile tra i processi figli) e dal minimo overhead per il cambio di contesto. Eventuali strutture a cui i thread potrebbero accedere contemporaneamente in scrittura sono protette da semafori mutex. La quantità di accessi in scrittura è comunque limitata, come l'eventuale durata degli stessi (solamente alla registrazione di un nuovo utente o all'aggiornamento del punteggio di un utente); l'overhead derivante dal dover attendere eventuali altri thread che bloccano i mutex è quindi basso.

Il server carica all'inizio del programma le uniche strutture dati memorizzate permanentemente tramite file. La lettura dei file è senza dubbio uno dei processi più lenti, ma grazie a questo accorgimento essa viene eseguita una sola volta e quando gli utenti non sono ancora collegati al server. Altre eventuali scritture su file avvengono solo quando un client si disconnette dal server. La struttura dei file è la seguente:

- nicknames.txt conserva i nomi degli utenti
- quiz.txt conserva i nomi dei quiz, le relative domande e risposte, formattato nel seguente modo
  THEME %numero_tema - %nome_tema
  %domanda? - %risposta
- score.txt conserva i risultati degli utenti nei vari quiz, formattato nel seguente modo
  %nome_utente %numero_tema %punteggio

Client

Il client è una semplice applicazione che invia poche informazioni: solamente il nickname, la scelta del tema e le risposte. La maggior parte della complessità è spostata sul server, come i controlli sui *nicknames*, che devono essere unici e di un massimo numero di caratteri prefissato. Il client fa solo due controlli:

- il primo sul range delle scelte che l'utente può fare: questo range viene comunicato inizialmente dal server, sarebbe stato quindi inutile reinviare un'eventuale scelta fuori dal range stesso; perciò, questo controllo è facilmente implementabile lato client;
- il secondo controllo invece riguarda il comando endquiz: il comando costringe il client a chiudere la connessione con il server e a ricominciare scegliendo un altro nickname, perciò quest'ultimo controllo risulta necessario in quanto modifica il normale flusso del processo client e questo lo porterebbe a desincronizzarsi dal processo server.

Strutture Dati

Le strutture dati del quiz sono tre liste: una per gli utenti, una per i temi ed una per i punteggi. Le liste sono implementate tramite generiche strutture *node* contenenti al loro interno anche un puntatore a *void*: questa scelta implementativa permette di poter utilizzare una struttura *node* senza dover definire una struttura specifica per ogni tipo di dato da memorizzare.

Questo rende molto facile i controlli su eventuali nickname non unici o nel caso in cui un client cerchi di scegliere un quiz a cui ha già partecipato: basterà infatti scorrere la relativa lista e inviare al client la risposta. Inoltre, sono implementate alcune funzioni per la gestione delle liste, come l'aggiunta di un nodo in testa, in coda o dopo un nodo specifico, e la rimozione di un nodo.

In questo modo è facile anche realizzare liste di liste, come nel caso della struttura *theme*: al suo interno è presente, oltre al nome del tema e l'id corrispondente, anche una lista di nodi che memorizzano nel campo *data* le strutture *question*, le quali hanno due campi *char[]* per memorizzare ogni domanda e la relativa risposta. Questo permette, una volta scelto il tema, di poter scorrere facilmente le domande una alla volta, senza accedere a file o dover scorrere nuovamente tutta la lista.

Le tre liste della struttura quiz sono inizializzate all'accensione del server, accedendo ai vari file che contengono le informazioni. Questo presuppone che tali file vengano mantenuti in uno stato consistente e che quindi vengano modificati solo dal server o da un eventuale amministratore che sappia come farlo mantenendo la formattazione e la coerenza fra di essi. Per esempio, il server assume che se esiste un punteggio per un utente nel file score.txt, quell'utente sia stato precedentemente caricato e trovato nel file nicknames.txt, stessa cosa per quanto riguarda i temi.

I nodi vengono inseriti nella struttura relativa ai punteggi in ordine decrescente, prima secondo l'id del tema (non secondo uno specifico ordine, ma per raggruppare i punteggi relativi allo stesso tema) e poi secondo il punteggio. Questo permette di stampare facilmente, con il comando showscore, la lista dei punteggi in ordine decrescente senza dover ricorrere ad una prima visita in lista per recuperarli, appoggiarsi quindi ad una struttura dati temporanea e procedere solo successivamente con una stampa in ordine.

Le strutture dati che implementano il quiz sono quindi abbastanza flessibili da poter supportare una quantità non prestabilita di quiz e di domande (per queste ultime è stato impostato un limite massimo di 5).

Tutti i parametri sono inoltre configurabili facilmente nel file config.h.