Documentazione del progetto di Reti Informatiche - Falaschi Tommaso [MAT.616097]

Il progetto realizza una applicazione client-server, dove in particolare sono presenti due client, realizzati in client.c e other.c, e un server. La comunicazione tra i client e il server è gestita con socket TCP, in quanto, nel nostro contesto, preferiamo garantire l'affidabilità rispetto alla rapidità della comunicazione che offre UDP. Lo scambio di messaggi tra i client e il server è realizzato grazie a due funzioni di utilità, invia e ricevi, all'interno delle quali viene prima inviata/ricevuta la lunghezza del messaggio e poi inviato/ricevuto il messaggio vero e proprio con formato text, dato che la comunicazione avverrà soltanto con lo scambio di stringhe. Questa implementazione sicuramente aumenta il traffico generato dovendo gestire una doppia send/recv ma permette di semplificare la gestione della comunicazione.

SERVER

Il server è implementato tramite I/O multiplexing, questo perché devono essere gestite in contemporanea lo STDIN, il socket di connessione e di comunicazione, e questa implementazione ci rende più semplice la gestione. Per il server sono disponibili soltanto i comandi start e stop. Fino a che non viene immesso il comando start verrà gestito soltanto il socket relativo allo STDIN. Quando viene invece difigitato il comando di stop il server termina la sua attività. Al momento della connessione dei client col server viene effettuato un breve handshake, che inizia con l'identificazione da parte del client, per verificare che la connessione venga stabilita nel modo corretto e per attribuire il ruolo del client all'interno del gioco. Gli identificatori dei socket sono posti in variabili apposite per facilitare l'utilizzo della funzionalità speciale. Superato l'handshake il server e i client comunicheranno tramite lo scambio di messaggi. In particolare, per il client giocatore (client.c), esso manderà al server i comandi da eseguire che verranno gestiti esclusivamente dal server tramite appositi gestori e funzioni di utilità. Il carico della gestione di eventuali comandi errati o situazioni di errore viene scaricato tutto sul server cercando di mantenere il più semplice possibile il lavoro del client. Questa scelta permette di non appesantire il client permettendo anche ai meno efficienti di poter usufruire del servizio. Di contro si ha che il carico, lasciato tutto al server, aumenterebbe all'aumentare dei client, oltre a un maggiore traffico generato dall'invio del comando e della risposta, il quale potrebbe essere evitato se i casi di errore venissero gestiti nel client. La comunicazione con il client aiutante (other.c), avviene invece nelle funzioni di gestione della funzionalità speciale, con semplice ricezione della parola immessa e invio di una parola in risposta. Il server si occupa anche dell'allocazione delle strutture dati per la gestione della partita utilizzando sia la memoria che i file di testo. Su file di testo è stato deciso di salvare le informazioni utili per il login di un utente, questo perché vogliamo che siano dati persistenti nel tempo anche a seguito di un guasto o un riavvio del server. L'accesso ai file di testo richiede un costo maggiore rispetto all'accesso in memoria, ma questo svantaggio non supera il vantaggio della persistenza, anche considerando che il numero di accessi è ridotto dal fatto che viene fatto soltanto accesso in scrittura per la registrazione e in lettura per il login. Le strutture dati principali sono la struct user_on, e la struct partita. La prima rappresenta un utente loggato, mentre la seconda una partita avviata da un utente. Per memorizzare le istanze di queste strutture sono state implementate due liste, questo per poter permettere in un futuro a più utenti di essere loggati contemporaneamente e poter avviare partite diverse. Nel caso particolare di questo progetto la cosa non è necessaria dato che si avrà un singolo utente loggato e una sola partita avviata. Gli elementi di queste liste vengono deallocati non appena un utente si disconnette o, nel caso della partita, la partita termina. L'uso della memoria in questo caso è necessario dato che vogliamo avere un accesso veloce alle strutture dati durante una partita

CLIENT

Anche il client è stato implementato con I/O multiplexing, facendo questa scelta per semplificare la gestione dei socket, e in modo tale che quando il server terminerà la sua attività tramite il comando stop verrà immediatamente disconnesso anche con una partita in corso. Principalmente nel client viene usato lo STDIN, nel quale verranno immessi i comandi da inviare al server. Quando il client si connette deve iniziare l'handshake col server identificandosi, ricevendo se tutto è andato correttamente, i comandi di partita. Successivamente, quando il giocatore immetterà un comando, questo verrà inviato al server e il client si metterà in attesa di una risposta che può rappresentare una stringa particolare che può permettere al client di gestire alcune casistiche internamente. Il giocatore potrà avviare una partita solamente se ha effettuato la registrazione e al successivo login verrà creata la struttura per l'utente loggato. A questo punto il giocatore potrà avviare una partita, con la creazione della struttura per la relativa partita. Se la partita termina questa verrà deallocata ed eventualmente allocata se ne verrà avviata un'altra.

OTHER

Il client aiutante è una versione semplificata del client giocatore. In questo caso non viene usato l'I/O multiplexing data la semplicità e la ridotta partecipazione alla partita da parte dell'aiutante. Anche in questo caso inizialmente viene iniziato un handshake dal client che deve indentificarsi. La differenza è che in questo caso, una volta terminato l'handshake il client aiutante si metterà in attesa di una stringa di attivazione da parte del server. Questo client non ha necessità di registrazione e login dato che non è un vero e proprio giocatore ma ricopre un ruolo marginale. Una volta entrato in azione questo client effettuerà un semplice ciclo nel quale attenderà di poter scrivere nello stdin, scriverà la sua parola che verrà inviata al server ricevendo dallo stesso la parola scritta dal giocatore e riniziando nuovamente il ciclo. Una volta che la funzionalità speciale terminerà il client aiutante si metterà nuovamente in attesa della stringa di attivazione da parte del server. È stata fatta questa scelta per permettere al client giocatore di poter nuovamente utilizzare l'aiuto in una eventuale nuova partita. Il client aiutante terminerà la sua attività nel caso in cui venga terminata l'attività del server.

FUNZIONALITA' A PIACERE

La funzione a piacere risiede in un nuovo comando guess utilizzabile soltanto dal client giocatore una sola volta per ogni partita. Questo comando permette di avviare un minigioco nel quale giocatore, per ottenere del tempo aggiuntivo, deve collaborare con un aiutante. In particolare, in ogni turno giocatore e aiutante scrivono una parola a testa senza sapere quella scritta dall'altro. Quando entrambi hanno scritto la parola viene inviata all'uno quella scritta dall'altro. L'obbiettivo è quello di scrivere entrambi la stessa parola cercando, in ogni turno, l'uno di avvicinarsi alla parola scritta dall'altro. Una volta che il giocatore digita il comando guess, il server, dopo aver controllato che il giocatore possa effettivamente attivare la funzionalità speciale, contatta l'aiutante inviando una stringa specifica per interrompere la sua attesa. Una volta che il client aiutante risponde vengono impostate delle variabili in modo tale che quello che si riceverà dal client giocatore fino al termine della funzionalità speciale venga usato da server come la parola scritta dal giocatore, e venga gestita tramite un apposito gestore che si occupa della ricezione della parola del client aiutante e trasmissione delle parole ai due client.