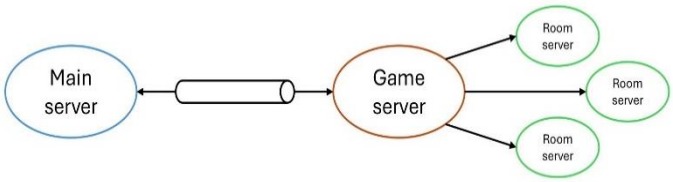
Alessio Simoncini
636010

Documentazione progetto Reti Informatiche A.A. 2023-2024

Server

Organizzazione e tipologia dei server



Il server è suddiviso in tre parti: server primario o di login (main server), server di gioco principale (game server) e server gestore di una partita (room server). Ho deciso di implementare questi tre moduli come processi, tutti gestiti con I/O multiplexing.

Main server È il processo che viene lanciato all'avvio dell'applicazione, viene gestito con io I/O multiplexing, grazie al quale controlla lo standard input in attesa di comandi da terminale, nuove connessioni, messaggi da client già connessi ed eventuali messaggi da game server. All'inserimento del comando `start <port>` crea, mediante fork, il processo game server con il quale mantiene per tutta la vita dell'applicazione due pipe per lo scambio di dati. Attualmente il game server non invia alcun messaggio al main server, ma la pipe e funzioni di utilità associate ([`send`|`recv`][`Code`|`Message`]`PIPE`) sono già presenti e potrebbero essere utilizzate per una nuova funzionalità che permette di tenere traccia dei punteggi e dei progressi di ogni utente.

All'inserimento del comando `stop` controlla che non ci siano client connessi ed invia, mediante pipe, il comando di stop al game server: solo dopo aver ricevuto la conferma della ricezione del messaggio il main server porta a termine la chiusura delle risorse utilizzate e termina a sua volta.

Riguardo all'interazione con i client è il main server è quello al quale si collegano i client all'avvio dell'applicazione: effettuano quindi la procedura di login o registrazione e rimangono collegati ad esso per tutta del loro ciclo di vita; il main server mantiene tutte le informazioni relative agli utenti su un file dedicato.

Game server È il processo dedicato alle richieste di nuove partite ed alla creazione delle stesse: ad ogni client viene comunicata la porta di questo server al termine della fase di autenticazione con il main server avvenuta con successo. Fornisce informazioni sulle stanze disponibili (codici e brevi descrizioni) ed accetta richieste di nuove partite ed eventuali annullamenti delle suddette richieste: mantiene le informazioni di tali utenti (username e file descriptor del socket) in un array che verrà poi passato al room server all'inizio della partita. Quando viene raggiunto un numero sufficiente di giocatori crea un processo room server per la gestione della partita e chiude i socket che verranno da questo momento gestiti dal room server.

Room server È un processo creato solo ed esclusivamente per la gestione di una partita e di conseguenza termina la sua esecuzione al termine della stessa.

L'implementazione di questo server mediante processi porta con sé lo svantaggio di un overhead considerevole all'avvio della partita: ho giudicato questo aspetto accettabile in quanto la parte dell'applicazione che necessita più delle altre di una buona responsività del server è la fase di gioco, di fatto successiva alla creazione del room server e quindi non influenzata dall'overhead della sua creazione. Inoltre la scelta dei processi si può rilevare vantaggiosa nel caso di partite che richiedono grandi risorse, ad esempio per il numero elevato di giocatori partecipanti o per la durata del gioco stesso.

Client

Il ciclo di vita del processo client è diviso in tre parti distinte, parallele ai tre tipi di server sopra elencati.

All'avvio dell'applicazione viene effettuata la connessione con il main server, con il quale è necessario effettuare l'autenticazione per ricevere la porta dove trovare il game server e accedere alla parte successiva.
Successivamente vengono mostrate a video le varie stanze disponibili e l'utente può mettersi in coda per una stanza a sua scelta, in attesa dell'arrivo di altri partecipanti; in questa fase, gestita con I/O multiplexing, il terminale non viene bloccato ma si permette l'annullamento della richiesta.
La fase di gioco è anch'essa gestita con I/O multiplexing, per permettere l'attesa di comandi da terminale ed in contemporanea controllare il socket del room server per la ricezione di messaggi da esso, siano comunicazioni di terminazione della partita per vittoria o gameover, ricezioni di token o messaggi da altri partecipanti.

Protocollo di trasporto
Ho deciso di utilizzare il protocollo TCP per tutte le comunicazioni dell'applicazione: la scelta è quasi obbligata per la parte di login e di richiesta partita, che necessitano la garanzia della correttezza delle informazioni scambiate, ma ho ritenuto che anche nella fase di gioco fosse fondamentale la correttezza delle informazioni a scapito della responsività.
Modalità di scambio dei dati
In tutta l'applicazione ho deciso di scambiare dati secondo text protocol, in quanto la maggioranza dei messaggi inviati consiste in stringhe che ben si adattano a questo formato; per mantenere l'uniformità del codice ed utilizzare le funzioni studiate per il text protocol anche nei casi di invio di interi o altre strutture ho deciso di utilizzare il text protocol.
Ho definito quattro funzioni per lo scambio di messaggi secondo text protocol: `sendCodeTCP` e `recvCodeTCP`, che inviano / ricevono un singolo byte, e `sendMessageTCP` e `recvMessageTCP`, che prima inviano / ricevono due byte contenenti la dimensione del messaggio e poi il messaggio stesso.
Protocollo di comunicazione
Tutti i protocolli di comunicazione per ogni tipo di richiesta sono elencati nel file `request_code.h`. Ogni protocollo è diverso a seconda delle necessità della richiesta, ma lo schema generale consiste nell'invio di un singolo byte via sendCodeTCP (codice di richiesta, sempre presente) che permette di identificare la richiesta, eventuali successivi messaggi e un codice di risposta, anch'esso un singolo byte. Si rimanda alla lettura del file `request_code.h` per una migliore trattazione dei protocolli di comunicazione.

Funzionalità a piacere
Come funzionalità a piacere ho introdotto la possibilità di inviare messaggi in tempo reale ad altri giocatori presenti nella partita. Il comando per inviare un messaggio ad un utente selezionato è:

```
> msg <username destinatario>
```

Inoltre è possibile inviare un messaggio a tutti i partecipanti specificando '`*`' come argomento del comando msg:

```
> msg *
```

All'avvio della partita il room server comunica ad ogni client gli username di tutti i partecipanti, ma non è possibile richiederli una seconda volta.
Questa meccanica, unita ad una curata progettazione delle stanze di gioco e un'analisi delle risorse concesse ai giocatori, può rivelarsi fondamentale per la risoluzione di un enigma e per la condivisione di informazioni e oggetti necessari in più punti della mappa.