Documentazione progetto di Reti Informatiche

Scelte implementative

Scelta del protocollo

Il protocollo di livello trasporto prescelto per la realizzazione è stato quello TCP. La motivazione principale è la garanzia della corretta trasmissione dei dati; questo a discapito della reattività di un protocollo meno affidabile come UDP.
Dato il contesto implementativo, il maggior traffico necessario al funzionamento di TCP non è un problema; viceversa, gli errori che possono derivare dall'uso del protocollo UDP potrebbero avere un impatto significativamente negativo sulla qualità del servizio.

I/O Multiplexing e Multithreading

Avendo valutato i diversi approcci, ho optato per la creazione di un server che utilizzando l'***I/O Multiplexing***, sia però in grado di servire contemporaneamente più client[1].
Il procedimento è il seguente: il server lavora come un semplice server *I/O Multiplexing*, quando un client fa una richiesta, esso toglie il socket id del client dal FD_SET master crea un **thread** e gli assegna il compito di gestire una sola richiesta di quel client: risolta la richiesta, il thread termina e il socket id del client torna nel FD_SET master.
Per rendere questa cosa possibile, la accept ha un timeout (nullo, in questo caso, per premiare la reattività, ma è possibile trovare dei compromessi), ciò permette al FD_SET master di essere modificato e, di conseguenza, la accept lavora su un FD_SET costantemente aggiornato: in poche parole abbiamo reso non bloccante la accept.

Text Protocols

Per quanto riguarda la trasmissione delle informazioni, la decisione più logica è stata quella di adottare protocolli testuali, in quanto offrono numerosi vantaggi in termini di interoperabilità e facilità d'implementazione. Questi protocolli testuali consentono una comunicazione efficiente e comprensibile tra i dispositivi e i sistemi coinvolti. Inoltre, la scelta di utilizzare protocolli basati su testo facilita la manutenzione e il debugging, oltre a garantire una maggiore trasparenza nella comunicazione dati.

Altri aspetti

Gestione della mutua esclusione

In generale, supponendo di avere più thread che operano contemporaneamente, la mutua esclusione è garantita per una serie di strutture globali: tavoli, tavoli_logged, prenotazioni, comande, listaThread.
Progettando il sistema, tuttavia, è nato l'ulteriore problema di garantire una mutua esclusione durante la prenotazione di un tavolo, non impedendo ad altri thread di agire in contemporaneo sulle strutture, in quanto ciò avviene in "2 tempi" intermediati da un'interazione dell'utente: la *find* e la *book*.
La risoluzione è coerente con le scelte implementative citate sopra: un thread rimarrà in attesa dell'utente, mentre altri thread possono agire sulla struttura. Sarà il primo thread a verificare, prima di effettuare modifiche, che ci siano i presupposti di accontentare l'utente; nel caso siano venuti a mancare, ripropone le nuove possibilità, agevolando un'altra scelta.

[1] Inteso come client generico del server, non strettamente di quello usato per eseguire prenotazione.

Gestione dei dati

I dati sono gestiti tramite strutture dinamiche.
Ad esempio, le prenotazioni sono gestite attraverso un array di puntatori – uno per ogni tavolo – dal quale parte una lista contenente tutte le prenotazioni per quel tavolo. Lo stesso vale per le comande.

La data all’interno della prenotazione è un campo di testo – scelto per comodità – sarebbe stato maggiormente corretto avere un campo data (ad esempio *time_t*).
L’utilità ai fini del progetto è valutare la disponibilità o meno del tavolo.

Sarebbe stato interessante salvare le informazioni – come ad esempio le comande effettuate – in memoria di massa (ad esempio un database relazionale), al fine di analizzare i dati inerenti alle preferenze degli utenti.

Note per l’utilizzo

I numeri dei tavoli vengono assegnati direttamente dal server al momento del collegamento, prima il tavolo 1, poi il 2 e così via fino ad un massimo di 16 (nMaxTd).
Viene ipotizzato che, una volta “acceso” un tavolo, rimane tale per tutta la durata in vita dell’applicativo server (immagino che venga acceso prima dell’apertura e, subito dopo la chiusura, spento).
Se ne viene scollegato uno, il successivo aperto prenderà il suo posto (classico esempio di riavvio per malfunzionamento del dispositivo).
È quindi consigliato, durante il test del progetto, effettuare prenotazioni su tavoli con numero basso, in modo che non serva tenere aperte molte finestre dedicate ai table device.

La maggior parte degli input sono controllati, lato server, per evitare segmentation fault o errori di altro genere; immagino però che, come giusto che sia in applicativi di tipo consumer, il software realizzato vada accompagnato ad un’interfaccia grafica la quale non consente input mal formattati.