Progetto di Reti Informatiche
Università di Pisa – Anno accademico 2022/23

Federico Giacomelli, matricola 578106

Presentazione

Il progetto è articolato su più directory.
Nella radice del progetto sono presenti 4 file in C: server, client, kitchen device e table device. Quindi ci sono i file di "exec2023.sh" che manda in esecuzione l'applicazione e il "makefile", utilizzato da "exec2023.sh", per compilare i file.
Ci sono poi le cartelle "header", contenente la definizione delle funzioni su file .h, "function", contenente l'implementazione delle funzioni su file .C e "storage" contenente i file di testo.
Tale organizzazione permette modularità e pulizia nel progetto, così da consentire modifiche nei file mirati evitando inoltre dimensioni eccessive nei file principali, in particolare il server.

Approccio

L'applicazione si basa sul modello Client-Server.
Il server è unico e ascolta sulla porta 4242, prefissata. Sono consentite invece molteplici connessioni da parte degli altri client, ognuno sulla sua porta di riferimento.
Non ho utilizzato la fork() vista la complicazione nella gestione di svariate connessioni multiple.

Il protocollo di trasporto è il TCP, visto il focus sull'affidabilità dell'informazione; del resto, da specifica, la velocità di trasferimento non era motivo di priorità.
Ciascun client instaura una connessione con il server, inviando un codice, e questo ne tiene traccia tramite vari array eventualmente impedendo un numero eccessivo di accessi.

Lo scambio di messaggi avviene tramite socket bloccanti, i quali vengono utilizzati in set così da utilizzarli una volta pronti. Ho assunto che i trasferimenti fossero sempre interi, ovvero la chiamata di send() e receive() permettono il corretto e completo trasferimento di dati.

Per ogni messaggio inviato dai client al server, questo filtra il tipo di comando, quindi invoca la funzione corrispondente. Il controllo sulla correttezza sintattica della comanda è lasciato ai client.

Server

È l'unico ad avere accesso alle strutture dati specifiche: include il file "utility.h" dove queste sono definite. In particolare, per i tavoli, le sale, le prenotazioni, i menù, le comande e le prenotazioni servite.
Nel file "server.c", si gestisce le connessioni con i client, un eventuale input e il filtraggio dei comandi ricevuti dai client. Tutte le funzioni, e relative variabili, sono sviluppate nei file appositi, ognuno nominato per l'utilità che ha. Solo il server accede a questi file; a tal proposito, ho implementato variabili globali per scambiare dati fra i vari file.

Qualsiasi operazione di I/O sul server viene mostrata nel suo terminale, sotto forma di comando sintetico e viene scritta nel file "logServer.txt". Questo non si cancella ad ogni avvio.

E' possibile inserire come input *stat* … oppure *stop*.
Nel primo caso, il server scorre il file "book_attive.txt" e in funzione dei parametri controlla o meno lo stato (che è un intero) di ciascuna prenotazione per decidere se stamparla o meno. In ogni caso non modifica il file. Nel secondo, invia un segnale di stop a tutti i client in quel momento collegati e chiude il socket di ascolto.

Inizialmente si inizializza la lista dei tavoli e sale, leggendo il file “tavoli.txt”.

Per il client, tiene traccia delle prenotazioni possibili in una lista che non viene salvata in locale.

Per il table device, il controllo sul codice lo fa scorrendo il file “book.txt” e cercando una corrispondenza. Il menù lo scorre dal file “menù.txt”, le comande le scrive sia una lista sia in locale su “book_attive.txt”.

Per il kitchen device, scorre ed eventualmente modifica il file “book_attive.txt” settando opportunamente l’intero corrispondente allo stato della comanda.

Client
Il client instaura una connessione con il server e si mette in attesa di un comando a scelta dell’utente. Nei limiti del possibile, riducendo anche il carico sul server, controlla la correttezza sintattica del comando e l’inserimento di data e ora per la prenotazione che non deve risultare precedente alla giornata odierna e deve essere compresa fra le 12-14 e le 20-22.

In ordine temporale, invia prima la *find* e poi la *book*.
Così mostra a schermo le possibili prenotazioni, quindi l’utente ne sceglie una e la invia; il server restituisce in caso di esito positivo il codice di prenotazione personale.

Per rendere più flessibili le prenotazioni, ho supposto che i tavoli si potessero accorpare con il solo di vincolo di farlo non cambiando sala. Ovvero, tutti i tavoli sono da 2 posti ma se il client chiede per 3 o più persone, il server accorpa più tavoli nella stessa sala, chiaramente non già prenotati. Se il numero di persone è dispari, approssima per eccesso. Ogni sala si ritiene piena se mancano sufficienti tavoli accorpabili e dunque la somma dei posti a sedere utili è inferiore alla richiesta. Nel caso la sala abbia più tavoli accorpabili, si scelgono i primi disponibili che soddisfano la richiesta e questi costituiscono l’unica scelta per la sala. Per ogni tavolo è indicato un codice e l’ubicazione, anche in caso di unione.

Table device
Il table device instaura una connessione con il server e si mette in attesa di un comando di prenotazione fintanto che l’utente non ne indica uno corretto.
Quindi viene “sbloccato” e questo può richiedere l’aiuto per i comandi, il menù, inserire una comanda o chiedere il conto; questo non può essere richiesto prima che tutte le comande siano andate in servizio.
Come nel caso precedente, si tende a ridurre il carico sul server e prima di inviare un comando controlla la correttezza sintattica dello stesso.

Kitchen device
Il kitchen device instaura una connessione con il server e:

- Mostra un * per ogni comanda che viene inviata al server.
- È in attesa di un input da parte dell’utente, quale che sia *take*, *show* o *ready*.

Nel caso di *take*, il server invierà un messaggio al table device che ha effettuato la comanda.
Nel caso di *ready* occorre indicare il numero di comanda seguito dai tavoli interessati.
Anche qui, svolge un controllo sintattico.

Nota
Con il comando “make clean” si rimuove anche il log e le informazioni sulle prenotazioni.
Con il comando “exec2023.sh”, oltre a compilare e mandare in esecuzione tutto, crea anche i file di log e di prenotazioni, qualora siano stati eliminati in funzione di un primo avvio.