DOCUMENTAZIONE PROGETTO RETI INFORMATICHE 2022/23

VISIONE DI INSIEME:

Nell'affrontare questo progetto sono state applicate le nozioni sulla programmazione distribuita in C viste a lezione, per la creazione delle varie applicazioni di rete è stato scelto un paradigma **Client-Server** e l'applicazione del protocollo **TCP**, sfruttando quindi la connessione tra i socket e l'affidabilità del protocollo.

Per la gestione contemporanea dei molteplici socket è stato impiegato l'**I/O Multiplexing**; sul processo server, per poter discriminare i vari tipi di processo client, è stato creato un file ("devices.txt") in cui si associa al file descriptor del socket il tipo di client connesso, comunicato da quest'ultimo in fase di connessione, quando un client si disconnette la riga corrispondente a esso nel file viene rimossa (si memorizzano infatti solo le connessioni attive, non la cronologia delle connessioni passate).

I dati utili del servizio sono stati prevalentemente salvati su **file** appositi per renderne possibile il recupero in caso di malfunzionamento di un processo.

Lo scambio di informazioni tra i processi avviene utilizzando sia il protocollo **text** che **binary**, il primo prevalentemente quando si ha scambio di messaggi in modalità verbosa, il secondo per trasmettere più agevolmente valori numerici.

Per la gestione delle richieste da parte del server è stato tenuto in conto il tipo di servizio offerto dall'applicazione e valutato possibile l'utilizzo di un **server iterativo,** in quanto non si attende un numero di richieste troppo elevato per la gestione delle prenotazioni da necessitare più processi dedicati, e si conosce staticamente il numero massimo di dispositivi interni (table e kitchen) rendendo possibile prevederne il flusso di richieste al server.

In generale la soluzione proposta non risulta resistente a un aumento esponenziale di connessioni, in quanto la potenza computazionale richiesta dal singolo processo server sarebbe troppo elevata e questo non riuscirebbe a gestire tutte le richieste accodate in tempi utili rischiando continuamente di mandare in overflow il buffer di sistema.
Tuttavia, potrebbe risultare possibile un'operazione di *scaling out*, per esempio con l'aumento di punti vendita e il conseguente aumento di clientela si potrebbe mantenere la funzionalità del servizio associando a ogni ristorante della catena un server dedicato.

GESTIONE CLIENT (cli.c):

Il processo subito dopo aver eseguito la primitiva connect esegue una send, comunicando il proprio tipo 'C'. Attende l'input della comanda da seguire:

- **find**: a seguito di controlli sull'input (ad esempio data futura o orario di apertura ristorante) costruisce il messaggio in protocollo text con la forma:

  "F <nome> <posti> <data> <ora>"

  (dove 'F' è un carattere usato dal server per diversificare l'operazione di find da quella di book) e lo invia al server. In risposta riceverà un valore numerico '0' se non c'è disponibilità per la richiesta eseguita, oppure la dimensione del messaggio di proposta dal server. Se quindi il valore ricevuto sarà diverso da '0' eseguirà una successiva recv per ottenere la disponibilità del ristorante e mostrarla all'utente.
- **book**: il comando risulterà valido solo se è stato eseguito almeno un comando *find* precedentemente. Consulta l'ultimo elenco di disponibilità arrivato dal server richiede una prenotazione al server tramite il messaggio text in forma:

  "B <data> <ora> <tavolo> <nome> <posti>"

  Il server controlla che la prenotazione sia ancora disponibile (un altro client potrebbe aver prenotato tra quando è arrivata la disponibilità ed è stato eseguito il comando book), la salva

nel file “reservation.txt” e comunica numericamente al client il codice di prenotazione, se questa è andata a buon fine, ‘0’ altrimenti.
- **esc**: chiude la connessione e termina il processo.

GESTIONE TABLE DEVICE (td.c)

Come gli altri processi comunica al server il proprio tipo (‘T’). Questo processo (come quello kitchen) ascolta contemporaneamente il socket connesso al server e lo standard input tramite I/O multiplexing.

- ***login***: prima di sbloccare il dispositivo viene richiesto il codice di prenotazione (questo avrà un valore ≥ 2000, per permettere al server di discriminare tra la procedura di login e l’invio di una nuova comanda dal table device) e viene comunicato al server che controlla il file “reservation.txt” per verificarlo. Il dispositivo riceverà il numero del tavolo se il codice è valido, un valore elevato (maggiore al numero massimo di tavoli) se il codice è già stato utilizzato su un altro dispositivo, ‘0’ se non esistono prenotazioni associate a esso.
- **menu**: mostra il menù, salvato su un file, ai commensali.
- **comanda**: utilizza la comanda in ingresso per andare ad aumentare le quantità per ogni piatto in una struttura dati apposita. Associa alla comanda un codice incrementale (com#) e lo stato di attesa. Aggiunge la comanda nel file unico “ordini$$$$.txt” (dove $$$$ rappresenta il codice di prenotazione). Invia al server prima la dimensione del messaggio, poi il messaggio con la comanda. Il server risponde con il messaggio “RICEVUTA” come conferma. Il server si occupa poi di notificare tutti i kd che è presente una nuova comanda in stato di attesa.
  Nel file “ordini$$$$.txt” dei vari tavoli, come nel file “ordinations.txt” del server, lo stato della comanda viene indicato dai caratteri ‘a’ (in attesa), ‘p’ (in preparazione) e ‘s’ (in servizio).
- **conto**: controlla nel file ordinazioni del tavolo che tutte le comande siano in stato di servizio, se lo sono calcola il conto sfruttando la struttura dati in memoria e lo mostra.

GESTIONE KITCHEN DEVICE (kd.c)

Come gli altri processi comunica al server il proprio tipo (‘K’). Come il table device utilizza I/O multiplexing.
Quando riceve dal server un valore >2000 lo interpreta come aggiornamento del numero di comande in attesa e stampa num_ricevuto-2000 ‘*’ (se =2000, non ci sono comande da accettare)

- **take**: se sono presenti comande in attesa, comunica al server, tramite il valore numerico ‘1’, che un cuoco accetta una comanda. In risposta riceve la dimensione del messaggio (<2000) e di seguito la comanda stessa mostrandola sul dispositivo. Salva tutte le comande accettate dal dispositivo in una struttura dati in memoria. Il server notifica tutti gli altri kitchen device che una comanda è stata accettata e il table device associato al tavolo che la sua comanda è in preparazione.
  Il server mantiene in memoria l’associazione tra il numero del tavolo (T#) e il file descriptor del socket associato a esso.
- **show**: mostra le comande nella struttura dati, quindi accettate, che siano nello stato di preparazione.
- **ready**: se la comanda esiste, comunica al server la dimensione del messaggio e successivamente il messaggio composto da “<tavolo> <codice comanda>” che identifica la comanda. Riceve dal server il valore ‘65535’ (2^{16}-1) come conferma di comanda in servizio. Il server notifica il td associato tavolo che la sua comanda è in servizio.

GESTIONE SERVER (server.c):

- **stat**: apre il file “ordinations.txt” e mostra le comande associate alla caratteristica richiesta: <tavolo>, <stato> o tutte se non specificata.
- **stop**: controlla dal file “ordinations.txt” che tutte le comande siano in servizio, se lo sono chiude la connessione con tutti i dispositivi ‘T’ e ‘K’, che avvieranno la procedura di chiusura.