

Progetto Reti Informatiche [Cod. 545II]

Visione d'insieme La struttura è stata progettata come un'applicazione **Client-Server**. Le comunicazioni fra i processi sono state regolate utilizzando I/O Multiplexing (per la facilità di utilizzo e per non creare eccessivo overhead per la generazione di processi figli), ed in secondo luogo grazie ad appositi protocolli utili a discriminare le azioni da eseguire. Tutti i client, una volta instaurato il canale di comunicazione, provvedono ad eseguire un **handshake** col server, che andrà a registrare in un file (*connections.txt*) la tipologia del client ('C', 'T' o 'K') unitamente al socket descriptor corrispondente. Nelle successive interazioni col server, questo sarà in grado di "riconoscere" il dispositivo ed interpreterà correttamente il **carattere indicante l'azione** da eseguire (e.g. {'C' + 'F'} → Client desidera eseguire una *find*).

Per registrare tutti i dati, è stato stabilito di sfruttare l'utilizzo di **file** appositi. Un'alternativa sarebbe potuta essere il salvataggio in strutture dati, ma a seguito di alcune valutazioni è stata prediletta la prima strada. In particolare, se da un lato la gestione con strutture dati avrebbe facilitato il lavoro in fase di sviluppo, la separazione dei dati dal programma stesso avrebbe reso il sistema più resistente ad eventuali guasti, garantendo un ripristino dei dati efficace. Inoltre, l'utilizzo di file potrebbe permettere di scalare l'applicativo con più facilità, di fronte ad un incremento di dati da dover registrare. A posteriori, avrei utilizzato un **ibrido** fra le due opzioni (e.g. per le connessioni bastava una struttura). Le comunicazioni sono state fatte in entrambe le modalità "TEXT" e "BINARY" (quest'ultima utilizzata solamente per trasmettere più agilmente campi numerici come le lunghezze delle stringhe da inviare). Comunicare in formato TEXT ha reso più lunghe le operazioni di parsing delle strutture dati.

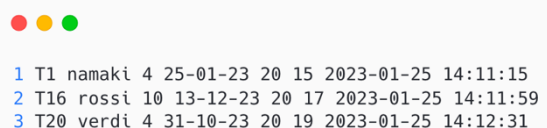
È stato scelto di realizzare un server **iterativo**, pensando che il carico di richieste di un ristorante non sia così elevato da richiedere più processi in esecuzione, senza doversi quindi preoccupare di accedere le risorse in mutua esclusione, ma accedendovi in maniera seriale. Riguardo alla **scalabilità**, la soluzione potrebbe scalare bene orizzontalmente, pensando di dedicare un server ad ogni ristorante coinvolto, così da ridistribuire il carico su più processi distinti.

1. Gestione Client Il dispositivo (come anche kitchen e table device) permette di svolgere le azioni richieste ed al contempo di essere sensibile ad eventuali segnali di stop del server grazie all'I/O Multiplexing. Si illustrano le scelte fatte per gestire le azioni da compiere:

- **find ['F']** → Dopo opportuni controlli di validità sui dati inseriti in input, il client invia al server la richiesta di prenotazione in modalità "TEXT", aspettando un riscontro positivo (proposte) o negativo (messaggio vuoto). Il server, da parte sua, effettua un parsing della richiesta ed elabora le proposte, che andrà a registrare su un file dedicato al client connesso (e.g. "P7.txt", dove 7 è il sd del client), inserendo anche la richiesta stessa per recuperarla alla conferma della prenotazione.
- **book ['B']** → Alla ricezione dell'opzione scelta, il server controlla l'esistenza della proposta scelta ed in caso affermativo registra la nuova prenotazione nel file "reservations.txt" con un id univoco incrementale, che sarà il codice della prenotazione.
- **esc** → chiude il socket di comunicazione e termina il processo.



```
daniel 4 13-03-23 20
1) T2 SALA1 CAMINO
2) T3 SALA1 TERRAZZA
3) T4 SALA2 -----
4) T5 SALA2 INGRESSO
5) T6 SALA2 -----
6) T7 SALA3 TERRAZZA
7) T8 SALA1 FINESTRA
8) T9 SALA1 -----
```




```
1 T1 namaki 4 25-01-23 20 15 2023-01-25 14:11:15
2 T16 rossi 10 13-12-23 20 17 2023-01-25 14:11:59
3 T20 verdi 4 31-10-23 20 19 2023-01-25 14:12:31
```

2. Gestione Table Device L'interazione con il dispositivo avrà inizio con un login in cui il cliente andrà ad inserire il proprio codice di prenotazione, di cui verrà certificata la validità (codice esistente, login

eseguito nello slot temporale previsto¹). In caso di esito positivo, andrà ad etichettare il td con il numero di tavolo legato al codice di prenotazione. È stato scelto questo approccio pensando che i table device vengano consegnati dai camerieri all'ingresso del ristorante, e che dunque possano assumere le qualità di tavoli diversi in timeframe distinti. Un altro motivo è stata la facilitazione che questa implementazione garantiva, consentendo di eseguire meno processi in fase di testing rispetto alla totalità dei tavoli presenti sulla piattaforma (20, per le ipotesi). Si illustrano le scelte fatte per gestire le azioni da compiere:

- **menu ['M']** → Il td contatta il server, il quale restituisce il menu integralmente (modalità "TEXT")
- **comanda ['C']** → Per ciascun piatto presente nella comanda viene fatto un primo controllo di sintassi sul client. Se questo viene superato l'ordine viene inviato al server, che controlla l'esistenza del piatto nel menù. Se un piatto all'interno di una comanda è invalido, viene ignorato. Questo ragionamento è stato applicato per evitare di far rimandare tutta la comanda integrale al cliente in caso di piatti errati. Si è scelto di delegare al td il compito di registrare il numero di comande inviate, che verrà comunicato al server al momento dell'invio di una nuova comanda in modo da distinguerle. Se è presente almeno un piatto valido, il server provvede ad aggiungere una nuova comanda nel file delle comande (*orders.txt*) e nel file degli ordini del td (e.g. O9, dove 9 è il file descriptor del td). Le comande si distinguono in 'K-' (in attesa), 'K[fd]' (in preparazione dal kitchen device fd) e 'KK' (in servizio). Alla ricezione della comanda il server andrà a segnalare la presenza di una nuova comanda a tutti i socket il cui device type registrato sarà 'K', conseguentemente restituirà al td lo stato delle sue comande.
- **conto ['R']** → Se tutte le comande relative al tavolo sono state poste in servizio (tutte avranno il campo 'KK'), allora il td potrà richiedere il conto. Quest'ultimo leggerà tutti gli ordini dal file relativo al td (O[fd].txt) e leggendo i prezzi da *menu.txt* calcolerà la spesa totale. Alla chiusura del pasto, tutte le comande relative al td verranno eliminate per evitare che nei pasti successivi si considerino anche le comande passate di altri clienti allo stesso tavolo.



```
-orders.txt
9 T9 com1 A1-1 K13
9 T9 com2 A1-1 K11
9 T9 com3 A2-2 S1-2 D1-3 K-
```

3. Gestione Kitchen Device Si illustrano le scelte fatte per gestire le azioni da compiere:

- **take ['T']** → Per andare a reperire la comanda con più priorità, il server selezionerà la prima riga trovata con 'K-', prendendo i dettagli della comanda e restituendoli al kd richiedente. Oltre a questo, il server andrà a modificare 'K-' inserendo 'K[fd]' per registrare il cambio di stato.
- **show ['S']** → Se sono presenti delle richieste prese in carico dal kd richiedente, allora il server leggendo il file *orders.txt* costruirà la stringa con le comande trovate restituendola al kd.
- **ready ['R']** → Dopo aver ricevuto la comanda, il server controllerà se la comanda era attualmente in preparazione e se questa era stata presa in carico dal kd richiedente; dunque, confermerà la richiesta ('K[fd]' → 'KK').

4. Gestione Server Si illustrano le scelte fatte per gestire le azioni da compiere:

- **stat** → Restituisce lo stato delle comande andando a elaborare il file *orders.txt*
- **stop** → Effettua un controllo di assenza di comande in attesa/preparazione nel file *orders.txt* e successivamente invia un byte (0) a tutti i socket connessi. Questi interpreteranno il messaggio avviando le procedure di chiusura, ed infine il server terminerà la propria esecuzione.

¹ Si suggerisce in fase di test di inserire prenotazioni relative al corrispondente orario di sistema attuale del calcolatore per poter provare l'esecuzione. Per impostare il fuso orario italiano (Debian 8-10) eseguire il comando: `sudo timedatectl set-timezone Europe/Rome`