Modalità di trasmissione dati

Nel progetto avviene l'interazione tra il server e dispositivi di differente natura tramite scambio di messaggi di richiesta e risposta.
Tutti i dati vengono trasmessi e ricevuti tramite socket TCP, utilizzando un buffer utente di dimensione `BUFFER_LEN 1024` byte (definita in `utils.h`) e strutture dati apposite per la richiesta e la risposta, in modo che non debba essere trasmessa la dimensione dei dati precededentemente al loro invio, poiché questi possono variare in base al tipo di operazione da eseguire.

> *Per il progetto in questione, la dimensione del buffer è sufficiente per trasmettere tutti i dati necessari.*

Indice

- Modalità di trasmissione dati
  - Indice
  - Connessione di un dispositivo al server
  - Richiesta dal client
  - Risposta dal server
  - Tipi di dato scambiati
    - 1. Operazione `FIND`
    - 2. Operazione `BOOK`
    - 3. Operazione `TAKE`
    - 4. Operazione `READY`
    - 5. Operazione `UNLOCK`
    - 6. Operazione `MENU`
    - 7. Operazione `COMANDA`
    - 8. Operazione `CONTO`
  - Messaggi dal server
    - 1. Operazione `STOP`
    - 2. Operazione `UPDATE`
    - 3. Operazione `INCREASE`
    - 4. Operazione `DECREASE`

Connessione di un dispositivo al server

Quando un dispositivo si connette al server, invia un messaggio di inizializzazione contenente il tipo di dispositivo che si è connesso (`K` per il *kitchen device*, `T` per il *table device*, `C` per il *client device*). Nel caso di *kitchen device*, il server invia in risposta il numero di comande attualmente in attesa.

Richiesta dal client

```
typedef struct {
    operazione op;
    char serializedBody[BUFFER_LEN - sizeof(operazione)];
} request_t;
```

La richiesta è composta da:

- **`op`** indica il tipo di richiesta fatta (enumeratore `operazione`).
- **`serializedBody`** contiene i dati necessari al server per eseguire la richiesta in formato serializzato, in quanto le strutture dati in esso contenute potrebbero variare a seconda della richiesta fatta.

Risposta dal server

```
typedef struct {
    int esito;
    char serializedBody[BUFFER_LEN - sizeof(int)];
} response_t;
```

La risposta è composta da:

- **`esito`** : intero che indica l'esito dell'operazione richiesta, la sua interpretazione dipende da tipo di richiesta fatta. In caso di errore, contiene il valore -1.
- **`serializedBody`** contiene eventuali dati di risposta del server, che sono deserializzati a seconda della richiesta fatta. Nel caso in cui l'operazione non sia andata a buon fine, contiene una stringa di errore.

Tipi di dato scambiati

A seconda del tipo di operazione, il contenuto di richiesta e risposta può variare, così come la sua interpretazione.
A priori, in caso di errore nell'elaborazione della richiesta, il server invia una risposta con `esito == -1` e **`serializedBody`** contenente una stringa informativa sull'errore.

1. Operazione `FIND`

- **Richiesta**:
  - **`op`**: `FIND`
  - **`serializedBody`**: dati della prenotazione da cercare (struttura `reservation_t`).
- **Risposta**:
  - **`esito`**: numero di tavoli disponibili.
  - **`serializedBody`**: lista delle prenotazioni (struttura `table_t`).

2. Operazione `BOOK`

- **Richiesta**:
  - **`op`**: `BOOK`
  - **`serializedBody`**: numero del tavolo scelto (tipo `int`).
- **Risposta**:
  - **`esito`**: 0.
  - **`serializedBody`**: dati della prenotazione effettuata (struttura `booking_t`).

3. Operazione `TAKE`

- **Richiesta**:
  - **`op`**: `TAKE`
  - **`serializedBody`**: vuoto.
- **Risposta**:
  - **`esito`**: 0.
  - **`serializedBody`**: dati della comanda presa in carico (struttura `comanda_t`).

4. Operazione `READY`

- **Richiesta**:

    - **`op`** : `READY`
    - **`serializedBody`** : dati per individuare la comanda da segnalare come pronta (*data*, *ora*, *numero di comanda* e *tavolo associato*).
- **Risposta**:
    - **`esito`** : 1 se c'è stato un aggiornamento nelle comande prese in carico, 0 in caso di successo.
    - **`serializedBody`** : lista delle comande prese in carico se sono cambiate (il tavolo non è più attivo o la comanda specificata non era stata presa in carico).

5. Operazione `UNLOCK`

- **Richiesta**:
    - **`op`** : `UNLOCK`
    - **`serializedBody`** : codice della prenotazione (tipo `bookingCode_t`).
- **Risposta**:
    - **`esito`** : 0.
    - **`serializedBody`** : vuoto.

6. Operazione `MENU`

- **Richiesta**:
    - **`op`** : `MENU`
    - **`serializedBody`** : vuoto.
- **Risposta**:
    - **`esito`** : numero di portate.
    - **`serializedBody`** : lista delle portate (struttura `menu_dish_t`).

7. Operazione `COMANDA`

- **Richiesta**:
    - **`op`** : `COMANDA`
    - **`serializedBody`** : lista dei piatti ordinati (struttura `comanda_dish_t`).
- **Risposta**:
    - **`esito`** : numero della comanda.
    - **`serializedBody`** : vuoto.

8. Operazione `CONTO`

- **Richiesta**:

    - **`op`** : `CONTO`
    - **`serializedBody`** : vuoto.
- **Risposta**:
    - **`esito`** : 1 se ci sono comande richieste in preparazione o in servizio, 0 se non se sono state inviate oppure sono ancora tutte in attesa.
    - **`serializedBody`** : lista delle voci che compongono il conto (struttura `subtotale_conto_t`), vuoto se non ce ne sono.

Messaggi dal server

Il server, inoltre, può inviare messaggi di notifica ai dispositivi connessi per segnalare l'aggiornamento di una comanda o di una prenotazione o la chiusura del ristorante.

```
typedef struct {
    server_operation_t header;
    char serializedBody[BUFFER_LEN - sizeof(int) - 1];
} server_message_t;
```

I messaggi inviati dal server sono composti da:

- **`header`** : tipo di messaggio inviato (enumeratore `server_operation_t`).
- **`serializedBody`** : dati del messaggio in formato serializzato, in quanto le strutture dati in esso contenute potrebbero variare a seconda del tipo di messaggio inviato.

1. Operazione `STOP`

- **Messaggio**:
    - **`header`** : `STOP`
    - **`serializedBody`** : vuoto.
- **Descrizione**: il server invia questo messaggio a tutti i dispositivi connessi per segnalare la chiusura del ristorante.

2. Operazione `UPDATE`

- **Messaggio**:
    - **`header`** : `UPDATE`
    - **`serializedBody`** : numero e stato della comanda aggiornata.

- **Descrizione**: il server invia questo messaggio allo specifico *table device* per segnalare l'aggiornamento dello stato di una comanda inviata.

3. Operazione `INCREASE`

- **Messaggio**:
  - **`header`**: `INCREASE`
  - **`serializedBody`**: vuoto.
- **Descrizione**: il server invia questo messaggio a ciascun *kitchen device* per segnalare l'incremento del numero di comande in attesa di essere accettate.

4. Operazione `DECREASE`

- **Messaggio**:
  - **`header`**: `DECREASE`
  - **`serializedBody`**: vuoto.
- **Descrizione**: il server invia questo messaggio a ciascun *kitchen device* per segnalare il decremento del numero di comande in attesa di essere accettate. Non viene inviato al *kitchen device* che ha preso in carico la comanda.