

Relazione progetto reti dei calcolatori Food Delivery



Docenti: Umberto Scafuri e Alessio Ferone

Studente: Eduardo Autore

Matricola: 0124001586

A.A. 2020/2021

Eduardo Autore

UNIVERSITA' DEGLI STUDI DI NAPOLI PARTHENOPE

Sommario

Descrizione del progetto	2
Descrizione e schemi dell'architettura	2
Descrizione e schemi del protocollo applicazione.....	3
Dettagli implementativi del client.....	6
Dettagli implementativi del rider	7
Dettagli sull'architettura server utilizzata.....	9
Dettagli implementativi del server.....	10
Dettagli implementativi del ristorante	13
Dettagli di esecuzione del client	15
Dettagli di esecuzione del rider (client)	18
Dettagli di esecuzione del server	20
Dettagli di esecuzione del Ristorante.....	22
Manuale dell'utente.....	24
Istruzioni per la compilazione	25
Istruzioni per l'esecuzione	25

Descrizione del progetto

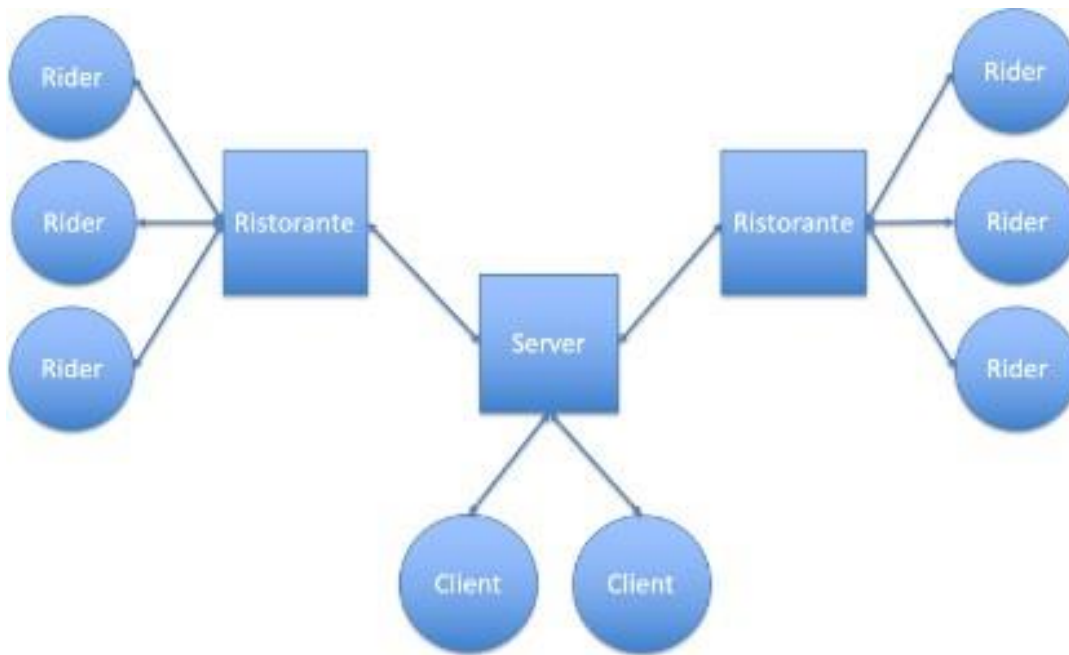
Il progetto consiste nel simulare un servizio di food delivery. Il client si collegherà al server, quest'ultimo gli consegnerà una lista di ristoranti così che il client possa scegliere uno di questi per connettersi. Il server richiederà al ristorante un menù che (una volta ricevuto) a sua volta invierà al client. Quest'ultimo sceglierà dei cibi o bevande dal menù, effettuerà un ordine e dopo averlo ultimato manderà la richiesta con l'ordine al server che a sua volta lo inoltrerà al ristorante, il quale verificherà la disponibilità di un rider. Il primo rider disponibile prenderà l'ordine e lo consegnerà al client che si identificherà da un ID e da un indirizzo. Una volta effettuata la consegna, il rider invierà una notifica dell'avvenuta consegna al ristorante che quindi notificherà anche il server.

Descrizione e schemi dell'architettura

Vi è un server di instradamento che gestisce il flusso di informazioni, queste ultime passano dal client al ristorante e dal ristorante al rider attraverso il server.

Il verso delle frecce indica il passaggio delle informazioni da una parte all'altra. I client, ovvero **rider** e **client**, sono rappresentati da un cerchio.

I server ovvero **il server di instradamento** e i **ristoranti**, sono rappresentati da quadrati.



Descrizione e schemi del protocollo applicazione

Il protocollo usato per il progetto è il TCP socket: prima di poter trasmettere dati è necessario stabilire una connessione tra mittente e destinatario. Questo modello è caratterizzato da due soggetti: un server che riceve richieste e fornisce servizi e un client che invia le richieste al server e riceve servizi da esso.

Sia il server di instradamento che il server ristorante sono **concorrenti (multithreading)** in quanto sono in grado di fornire servizi a più client contemporaneamente attraverso la creazione dinamica di un thread con la chiamata:

pthread_create(pthread_t nuovoThread, NULL, (void*)thread_i_function, &client_socket) . Quando un nuovo thread conclude il suo compito, il server continua a rimanere sempre in attesa di nuove richieste.

In questo programma sia il server di instradamento che il ristorante, fanno sia da server che da client:

- Server
 1. Come server:
 - Fornisce dei servizi ai client(thread_i_client):

invia su richiesta la lista di ristoranti connessi o il menù di uno specifico ristorante (precedentemente richiesti al ristorante in questione) oppure una notifica di consegna conclusa.

- Fornisce dei servizi ai ristoranti (`thread_i_restaurant`): registra le informazioni dei ristoranti connessi in una lista nel server per permettere la successiva connessione al ristorante se richiesta da un client.
- Fornisce dei servizi ai riders (`thread_i_rider`): invia ad un rider la lista dei ristoranti connessi disponibili per potersi connettere ad uno di essi.

2. Come client:

- Si connette al ristorante:
inoltra le richieste del client al ristorante ricevendone dei servizi ovvero riceve i menù e l'id del rider che sta effettuando la consegna.

• Ristorante

1. Come server:

- Fornisce dei servizi al server (`thread_i_server`):
invia su richiesta il proprio menù oppure l'ID del rider che sta eseguendo la consegna oppure una notifica di avvenuta consegna da parte di un rider.
- Fornisce dei servizi ai rider (`thread_i_rider`):
li mette in attesa in caso di mancanza di ordini client oppure permette la consegna al primo rider disponibile del primo ordine estratto da una coda di ordini.

2. Come client:

- Si connette al server:
il ristorante si connetterà al server solo la prima volta per farsi registrare, quindi per rendere disponibili le sue informazioni di connessione e per permettere al server di riconnettersi ad esso in un secondo momento.

• Client

- Si connette al server:
effettuerà una richiesta dei ristoranti disponibili , successivamente,
una volta ottenuto il menù del ristorante richiesto dal server,
effettuerà un ordine ed infine attenderà l'esito della consegna.
- Rider
 - Si connette al ristorante:
si connette al ristorante scelto e se ci saranno nuovi ordini, potrà
effettuare la consegna, altrimenti rimarrà in attesa di nuovi ordini.
 - Si connette al server: si conatterà solo per richiedere al server la
lista dei ristoranti.

Di seguito viene riportato uno schema del protocollo applicazione:

Mittente	Destinatario	Servizio Richiesto	Porta	Risposta del destinatario
Server	Client	Invia Scelta Lista ristoranti	8989	Scegli ristorante: Ristorante(id, nome, posizione))
Server	Ristorante	Richiesta menù	4000 + (1...MAX_PORT)	Invia menù
Server	Client	Invia menù	8989	Effettua_ordine (Num pietanze)
Server	Client	Richiesta utente: ID , nome, cognome e indirizzo	8989	Invio informazioni utente
Server	Ristorante	Invio scelta client e info	4000 + (1...MAX_PORT)	Ricezione info
Ristorante	Rider	Invio informazioni del client a cui consegnare	4000 + (1...MAX_PORT)	Invio ID rider
Rider	Ristorante	Notifica avvenuta consegna	4000 + (1...MAX_PORT)	Ricezione Info
Ristorante	Server	Notifica avvenuta consegna da parte del rider	4000 + (1...MAX_PORT)	Ricezione Info

Dettagli implementativi del client

Il client non svolge nessuna operazione particolarmente importante, si limita a leggere notifiche e dati mandati dal server.

La creazione della socket avviene tramite la funzione

```
client_socket = socket(AF_INET, SOCK_STREAM, 0);
```

AF_INET è la famiglia IP4, SOCK_STREAM è il tipo (canale bidirezionale per la trasmissione di pacchetti).

Successivamente si passa alla connessione col server tramite la funzione connect che permette la connessione della socket ad indirizzo e porta del server:

```
if (connect(client_socket, (SA*)&servaddr, sizeof(servaddr)) != 0)
```

Connesso il client al server, il client leggerà dapprima il size della lista di ristoranti e successivamente la lista dei ristoranti tramite la funzione “ricevi_lista_ristoranti” che userà size volte la funzione “ricevi_info_ristorante”.

Quest’ultima richiamerà per tre volte la funzione

```
FullRead(socket, buffer, sizeof(buffer));
```

che usa la system call read e permette la lettura di tutti i byte richiesti e che quindi si bloccherà fin quando non avrà ricevuto tutti i dati dei vari ristoranti.

```
/* Ricevi informazioni dal ristorante */
Ristorante *ricevi_info_ristorante(int socket){
    /* Alloco un nuovo ristorante */
    Ristorante *ristorante_ricevuto = (Ristorante *)malloc(sizeof(Ristorante));

    char nome_ristorante[MAX_NOME_RISTORANTE];
    char posizione_ristorante[MAX_POSIZIONE_RISTORANTE];
    char id_lettura[10];

    /* Riceve informazioni dal ristorante da memorizzare nella lista */
    FullRead(socket, nome_ristorante, sizeof(nome_ristorante));
    FullRead(socket, posizione_ristorante, sizeof(posizione_ristorante));
    FullRead(socket, id_lettura, sizeof(id_lettura));

    strcpy(ristorante_ricevuto->nome, nome_ristorante);
    strcpy(ristorante_ricevuto->posizione, posizione_ristorante);

    ristorante_ricevuto->id = atoi(id_lettura);

    return ristorante_ricevuto;
}
```

```

llist *ricevi_lista_ristoranti(int socket){
    char sizeList[100];
    FullRead(socket, sizeList, sizeof(sizeList));
    int size_list = atoi(sizeList);

    if(size_list == 0){
        printf("Nessun ristorante disponibile\n\n");
    }
    else{
        llist *list_restaurant = llist_create(NULL);
        /* in base alla dimensione della lista verranno ricevuti sizeList ristoranti (es: sizeList = 2 = 2 chiamate a "ricevi_info_ristorante") */
        for(int i = 0; i < size_list; i++){
            Ristorante *nuovoRistorante_caricato = ricevi_info_ristorante(socket);
            nuovoRistorante_caricato->id_scelta = size_list-i;
            llist_push(list_restaurant, nuovoRistorante_caricato);
        }
        return list_restaurant;
    }
}

return NULL;
}

```

Dopo aver effettuato una scelta, con la funzione FullWrite si invia la scelta al server e nello specifico con la funzione “invia_info_ristorante” che invia con tre FullWrite le informazioni del ristorante al server:

`FullWrite(socket, buffer, sizeof(buffer));`

che usa la system call write e permette la scrittura di tutti i byte e che quindi si bloccherà fin quando non avrà inviato tutti i dati del ristorante.

Successivamente ci sarà uno scambio di informazioni attraverso le funzioni FullRead e FullWrite.

Con la FullWrite verrà inviato un nuovo ordine contenente una coda di pietanze, l’ID dell’ordine ed il conto. Verranno fornite le informazioni del client(ID, nome, cognome e posizione) mentre con al FullRead si riceveranno informazioni come l’ID del rider che ha preso in carico la consegna e la notifica di avvenuta consegna da parte del rider.

Dettagli implementativi del rider

Il rider effettuerà le stesse funzioni del client, quindi creerà la socket e con la connect proverà a connettersi con il server, ricevuta la lista di ristoranti da quest’ultimo, proverà a connettersi con il ristorante scelto.

Dopo aver confermato la propria disponibilità (inviando una conferma al ristorante), leggerà le varie informazioni necessarie per la consegna, ricevute dal ristorante (Ordine con ID ed Utente con ID) con le funzioni “ricevi_ordine” e “ricevi_info_utente” che utilizzeranno la funzione

`FullRead(socket, buffer, sizeof(buffer));`


```

Ordine *ricevi_ordine(int socket){

    Ordine *nuovoOrdine_utente = (Ordine *)malloc(sizeof(Ordine));
    char id_ricevuto[10];
    char conto_ricevuto[317];

    int id_ordine;
    double conto;
    queue *carrello_utente = createQueue(sizeof(Ordinazioni *));

    /* Riceve l'ID dell'ordine creato */
    FullRead(socket,id_ricevuto,sizeof(id_ricevuto));
    id_ordine = atoi(id_ricevuto);

    /* Riceve il conto */
    FullRead(socket,conto_ricevuto,sizeof(conto_ricevuto));
    conto = atof(conto_ricevuto);

    /* Riceve il carrello dell'ordine dell'utente */
    carrello_utente = ricevi_ordinazioni(socket);

    nuovoOrdine_utente->id_ordine = id_ordine;
    nuovoOrdine_utente->conto = conto;
    nuovoOrdine_utente->queue_pietanze_ordinate = carrello_utente;

    return nuovoOrdine_utente;
}

```

```

Utente *ricevi_info_utente(int socket){

    Utente *nuovoUtente = (Utente *)malloc(sizeof(Utente));
    char id_client_ricevuto[10];
    char nome_utente[MAX_NOME_UTENTE];
    char cognome_utente[MAX_COGNOME_UTENTE];
    char posizione_utente[MAX_POSIZIONE_UTENTE];

    /* Riceve l'id del client */
    FullRead(socket,id_client_ricevuto,sizeof(id_client_ricevuto));
    int id_client = atoi(id_client_ricevuto);
    /* Riceve il nome dell'utente */
    FullRead(socket,nome_utente,sizeof(nome_utente));
    /* Riceve il cognome dell'utente */
    FullRead(socket,cognome_utente,sizeof(cognome_utente));
    /* Riceve la posizione dell'utente al server */
    FullRead(socket,posizione_utente,sizeof(posizione_utente));

    nuovoUtente->id = id_client;
    strcpy(nuovoUtente->nome,nome_utente);
    strcpy(nuovoUtente->cognome,cognome_utente);
    strcpy(nuovoUtente->posizione,posizione_utente);

    return nuovoUtente;
}

```

Successivamente inoltrerà al ristorante i propri dati con ID. Il ristorante poi inoltrerà questi dati al server che a sua volta verranno inoltrati al client.

Infine il rider invierà una notifica di avvenuta consegna(carattere 's') al ristorante dopo un certo tempo (appena effettuata la consegna).

Dettagli sull'architettura server utilizzata

Per gestire più connessioni concorrentemente , utilizziamo un un'architettura server di tipo multi-thread. Un thread è un singolo flusso di esecuzione indipendente all'interno di un processo, che lo scheduler può fare eseguire separatamente e concorrentemente con il resto del processo.

- Vantaggi

Gli svantaggi dell'utilizzare i processi (fork()) rispetto ai thread in un server sono legati principalmente al costo elevato, poiché ogni fork che gestisce una nuova connessione crea una nuova pagina di memoria con una copia esatta del processo padre PPID e quindi con un elevato spreco di memoria se il server riceve molte connessioni contemporaneamente ed una minore interoperabilità tra i processi poiché ogni processo possiede una propria area

in memoria (id, codice, dati, stack, risorse, stato CPU) e quindi per comunicare tra di loro è necessario utilizzare una forma di interprocess communication (IPC) ovvero le pipe (anonime o FIFO).

Oltre a questo la commutazione tra thread (cambio di contesto) è più veloce rispetto alla commutazione tra processi poiché l'uso di thread suddivide lo stato di un processo in due parti

- Lo stato della risorsa che resta con il processo
- Lo stato della CPU che è associato con il thread

Quindi possiamo lavorare sulla stessa area di memoria del processo (stato della risorsa) ma con più stati della CPU indipendenti che commutano più velocemente (Quindi più **thread control block** che puntano allo stesso **Process control block** condividendo il proprio spazio di indirizzamento).

Riduciamo gli overhead di: creazione e terminazione thread, comunicazione e cambiamento di contesto.

Inoltre sulle attuali architetture multiprocessore possiamo eseguire più thread in parallelo.

- Svantaggi

La condivisione delle risorse accentua il pericolo di interferenza quindi gli accessi concorrenti devono essere sincronizzati di modo da evitare interferenze (thread safeness). Quindi dobbiamo regolare l'accesso alle risorse condivise da parte dei thread tramite un apposito meccanismo di sincronizzazione (Semafori, mutex, monitor, variabili condizione) così da avere un accesso pulito in mutua esclusione (atomico) a queste risorse.

Nel progetto sono stati utilizzati dei semafori binari per l'accesso alle variabili condivise (code, liste etc...) e dei semafori contatori per la gestione dei rider in attesa, quindi quante più richieste (ordini) vengono effettuate, tante più sono le signal (sem_post) che vengono usate per sbloccare i thread in attesa che gestiscono i rider wait (sem_wait).

Dettagli implementativi del server

Il server legge le richieste dal client e gli dà delle risposte. Successivamente inoltra le informazioni al ristorante.

Il server necessita della funzione bind, una system call per associare il proprio indirizzo e la propria porta (su cui esso aprirà le proprie comunicazioni) ai client:

```
if ((bind(server_socket, (SA *) &servaddr, sizeof(servaddr))) != 0)
```

Successivamente bisogna mettere il server in ascolto (in attesa di connessioni) tramite la system call listen :

```
if ((listen(sockfd, 1000)) != 0)
```

Si fa partire un ciclo infinito con all'interno la system call accept per accettare le nuove connessioni, appena disponibili

Quando arriva la connessione la si deve accettare tramite appunto la accept:

```
connfd = accept(server_socket, (SA *) &client_addr, &len);
```

Appena viene accettata una connessione, viene letto con una FullRead un carattere da cui è possibile ricavare la tipologia di client che sta richiedendo la connessione (ristorante = 1, client = 2, rider = 3).

Ricavata la tipologia di client, viene creato dinamicamente un thread:

```
pthread_t *thread = (pthread_t *)malloc(sizeof(pthread_t));
```

Creato il thread, a seconda della tipologia di client, viene lanciata pthread_create() che crea e lancia il thread e l'opportuna funzione per gestire il suddetto client:

```
pthread_create(&thread, NULL, (void *)thread_i_function,  
&client_socket)
```

```

while(1){
    printf("Server Delivery food in attesa di connessioni...\n\n");
    /* Accetta una nuova connessione */
    client_socket_i = (int *)malloc(sizeof(int));
    *client_socket_i = accept_new_connection(server_socket);
    int client_socket = *client_socket_i;
    free(client_socket_i);

    /* Riceve un intero che identifica la tipologia di client che si vuole connettere al server */
    char ricevline_info_client[1];
    FullRead(client_socket, ricevline_info_client, sizeof(ricevline_info_client));
    int tipo_client = atoi(ricevline_info_client);

    /* Creo un nuovo thread per ogni connessione accettata */
    thread_i = (pthread_t *)malloc(sizeof(pthread_t));

    if(tipo_client == RISTORANTE_ID){
        /* inserisco nella coda di ristoranti un nuovo ristorante */
        sem_wait(&S1.mutex_queue1);
        enqueue(S1.queue_client1, (int *) &client_socket);
        sem_post(&S1.mutex_queue1);

        /* creo e lancio un thread per gestire la connessione da parte del ristorante */
        pthread_create(thread_i, NULL, (void *)thread_i_restaurant, &client_socket);
    }
    else if (tipo_client == CLIENT_ID){
        /* inserisco nella coda dei client un nuovo client */
        sem_wait(&S2.mutex_queue2);
        enqueue(S2.queue_client2, (int *) &client_socket);
        sem_post(&S2.mutex_queue2);

        /* creo e lancio un thread per gestire la connessione da parte del client */
        pthread_create(thread_i, NULL, (void *)thread_i_client, &client_socket);
    }
}

else if (tipo_client == RIDER_ID){
    /* inserisco nella coda dei client un nuovo client */
    sem_wait(&S3.mutex_queue3);
    enqueue(S3.queue_client3, (int *) &client_socket);
    sem_post(&S3.mutex_queue3);

    /* creo e lancio un thread per gestire la connessione da parte del client */
    pthread_create(thread_i, NULL, (void *)thread_i_rider, &client_socket);
}
}

```

1. Thread_i_restaurant

Viene creato quando un ristorante tenta di connettersi e viene gestito inserendo i dati (struttura Ristorante) ricevuti tramite FullRead del ristorante connesso in mutua esclusione in una lista condivisa di ristoranti.

2. Thread_i_client

Viene creato quando un client tenta di connettersi e viene gestito inviando tramite FullWrite la lista di ristoranti disponibili, successivamente aspettando un ristorante scelto dal client per poi connettersi al suddetto ristorante creando una seconda socket di connessione. Sulla socket di connessione del ristorante leggerà con delle FullRead il menù del ristorante, invierà il menù al client (FullWrite) e attenderà dal client un ordine con le informazioni client

(FullRead), inoltrerà queste informazioni al ristorante e attenderà la conferma di consegna presa in carico da un rider ed infine la notifica di fine consegna da inoltrare al client

3. Thread_i_rider

Viene creato quando un rider tenta di connettersi e viene gestito inviando tramite FullWrite la lista di ristoranti disponibili così da permettere al rider di sceglierne uno.

Dettagli implementativi del ristorante

Il ristorante legge le richieste dal server , inoltra ai riders le risposte e viceversa, legge le richieste dai riders ed inoltra al server le risposte. Dato che dopo la connessione al server per registrarsi, deve ricevere delle richieste in un secondo momento, la sua struttura è analoga a quella del server.

Il ristorante necessita della funzione bind , una system call per associare il proprio indirizzo e la propria porta (su cui esso aprirà le proprie comunicazioni) ai client:

```
if ((bind(server_socket, (SA *) &servaddr, sizeof(servaddr))) != 0)
```

Successivamente bisogna mettere il server in ascolto (in attesa di connessioni) tramite la system call listen :

```
if ((listen(sockfd, 1000)) != 0)
```

Si fa partire un ciclo infinito con all'interno la system call accept per accettare le nuove connessioni, appena disponibili

Quando arriva la connessione la si deve accettare tramite appunto la accept:

```
connfd = accept(server_socket, (SA *) &client_addr, &len);
```

Appena viene accettata una connessione, viene letto con una FullRead un carattere da cui è possibile ricavare la tipologia di client che sta richiedendo la connessione (server = 0, rider = 3).

Ricavata la tipologia di client, viene creato dinamicamente un thread:

```
pthread_t *thread = (pthread_t *)malloc(sizeof(pthread_t));
```

Creato il thread, a seconda della tipologia di client, viene lanciata pthread_create() che crea e lancia il thread e l'opportuna funzione per gestire il suddetto client:

```
pthread_create(&nuovoThread, NULL, (void *)thread_i_function,
&client_socket)
```

```
while(1){
    printf("\n\nRistorante [%s] in attesa di connessioni sulla porta [%d] ...\n\n", nuovoRistorante->nome, porta_temp);

    /* Accetta una nuova connessione */
    client_socket_i = (int *)malloc(sizeof(int));
    *client_socket_i = accept_new_connection(server_restaurant_socket);
    int client_socket = *client_socket_i;
    free(client_socket_i);

    /* Riceve un intero che identifica la tipologia di client che si vuole connettere al ristorante */
    char ricevline_info_client[1];
    FullRead(client_socket, ricevline_info_client, sizeof(ricevline_info_client));
    int tipo_client = atoi(ricevline_info_client);

    /* Creo un nuovo thread per ogni connessione accettata */
    thread_i = (pthread_t *)malloc(sizeof(pthread_t));

    if(tipo_client == SERVER_ID){
        /* inserisco una nuova connessione nella coda di connessioni server */
        sem_wait(&S0.mutex_queue0);
        enqueue(S0.queue_client0, (int *) &client_socket);
        sem_post(&S0.mutex_queue0);

        /* creo e lancio un thread per gestire la connessione da parte del ristorante */
        pthread_create(thread_i, NULL, (void *)thread_i_server, &client_socket);
    }
    else if (tipo_client == RIDER_ID){
        /* inserisco nella coda dei Riders un nuovo Rider (client) */
        sem_wait(&S3.mutex_queue3);
        enqueue(S3.queue_client3, (int *) &client_socket);
        sem_post(&S3.mutex_queue3);

        /* creo e lancio un thread per gestire la connessione da parte del client */
        pthread_create(thread_i, NULL, (void *)thread_i_rider, &client_socket);
    }
}
```

1. Thread_i_server

Viene creato quando il server tenta di connettersi e viene gestito inviando con delle FullWrite il menù al server, successivamente attendendo le informazioni sull'ordine da leggere con delle FullRead. Ricevute queste informazioni, il thread che gestisce il server attraverso il meccanismo di sincronizzazione gestito con dei semafori contatori, attiva un rider ad ogni nuova richiesta(Ordine), ovvero con una `sem_post(&sem_assegna_riders)`, successivamente attende la conferma da parte di un rider con una `sem_wait(&sem_conferma)`. Dopo aver ricevuto la conferma ed essere stato sbloccato, invia con delle FullWrite le informazioni del rider al server e riceve con delle FullRead le informazioni del client che ha effettuato la richiesta. Dopo aver reso tali informazioni al rider che ha preso in carico la consegna, riattiva il rider con `sem_post(sem_attesa_id)` ed attende il completamento

della consegna con `sem_wait(&sem_consegna_completata)` . Una volta sbloccato questo ultimo semaforo, verrà inviata la notifica di completamento consegna al server.

2. Thread_i_rider

Viene creato quando un rider tenta di connettersi e viene gestito rimanendo in attesa su `sem_wait(&sem_assegna_riders)` finchè non viene sbloccato dal ristorante dopo l'arrivo di una nuova richiesta. Una volta che si è preso in carico la nuova richiesta, invia la conferma al ristorante tramite `sem_post(&sem_conferma)` ed attende su `sem_wait(&sem_attesa_id)` che gli vengano fornite le informazioni del client da ristorante e che le sue informazioni(ID rider) vengano inviate dal ristorante al server. Una volta terminati gli scambi di informazioni, il ristorante riattiverà il rider `sem_post(&sem_attesa_id)` ed il rider potrà eseguire la consegna con le informazioni (utente e ordine) ricevute.

Terminata la consegna , invierà una notifica al ristorante risvegliandolo con `sem_post(&sem_consegna_completata)` .

Dettagli di esecuzione del client

Quando il programma client verrà lanciato, l'utente dovrà inserire un **nome**, un **cognome**, una **posizione**(importante per effettuare l'ordine) ed un **saldo** necessario per acquistare cibi e bevande, in più verrà generato un **ID** .Successivamente il client effettuerà una connessione con il server di instradamento e se quest'ultima avrà successo, il client rimarrà in attesa che il server gli invii la lista di ristoranti disponibili in quel momento


```
eduardo@eduardo-VirtualBox: ~/Documenti/Università_Parthenop...
eduardo@eduardo-VirtualBox:~/Documenti/Università_Parthenope/Reti di Calcolatori
/Progetto_RETI_Eduardo_Autore_0124001586/Codice/client$ ./client

-----INFORMAZIONI UTENTE-----
Inserisci il tuo nome: Giovanni
Inserisci il tuo cognome: Rossi
Inserisci la tua posizione: Via Luca Giordano 38
Inserisci il tuo saldo: 3200
```

```
*****FINE INFORMAZIONI UTENTE*****
*****UTENTE*****
ID: 837300382
Nome: Giovanni
Cognome Rossi
Posizione: Via Luca Giordano 38
Saldo: 3200.00
*****

[Canonical name]: 127.0.0.1
[Address type]: IPv4
[Address]: 127.0.0.1

server: 127.0.0.1 port: 8989
Fri Nov 5 16:18:56 2021 : Sei il [client] numero: [2]

1) ID: 638178286 - Nome Ristorante: Sorbillo - Posizione: Via Tribunali 32
2) ID: 260865843 - Nome Ristorante: L'Antica pizzeria Da Michele - Posizione: Via Cesare Sersale, 1
3) ID: 587557654 - Nome Ristorante: 50 Kalo' - Posizione: Piazza Sannazzaro

Scegli l'id del ristorante da cui vuoi ordinare: 
```

Scelto il ristorante, verrà inviata la scelta effettuata al server (ristorante: id, nome, posizione) mentre il client rimarrà in attesa del menù del ristorante scelto. Il server si conatterà con il ristorante richiesto dal client e se la connessione avrà successo, richiederà al ristorante designato il menù, il quale una volta ricevuto dal ristorante, verrà inoltrato dal server al client e quest' ultimo si sbloccherà dando all' utente la possibilità di effettuare un nuovo ordine (coda di cibi, bevande e costi).

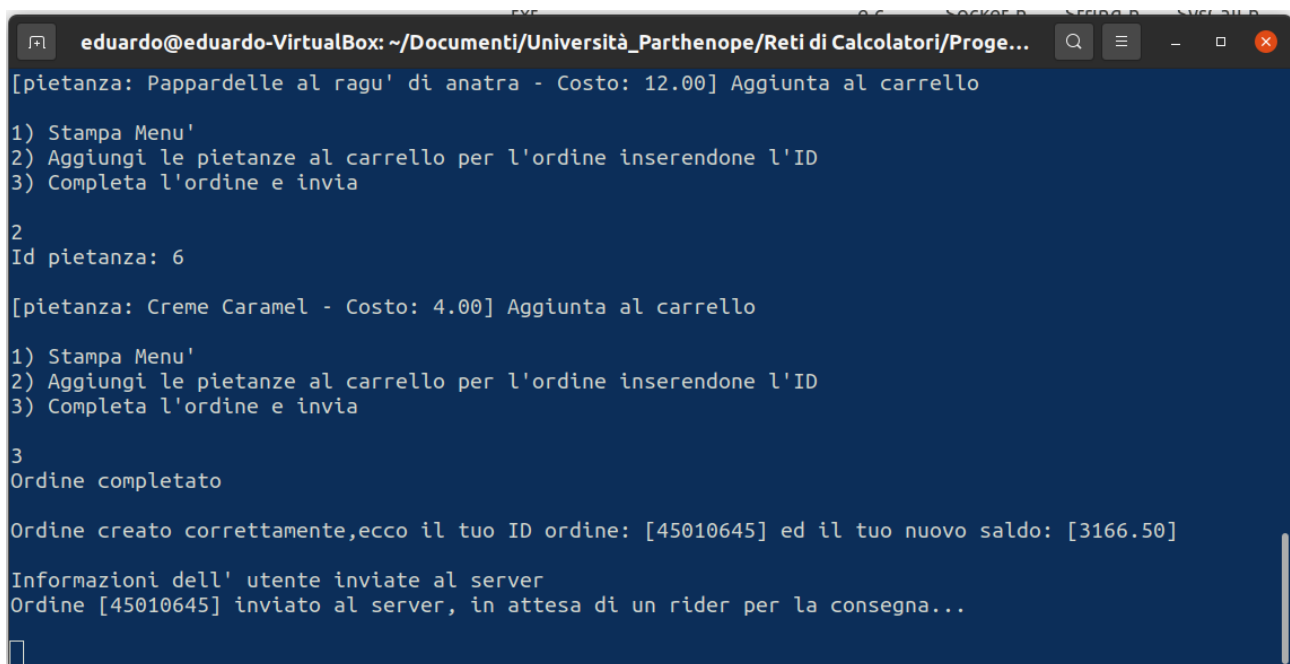
```
1) ID: 638178286 - Nome Ristorante: Sorbillo - Posizione: Via Tribunali 32
2) ID: 260865843 - Nome Ristorante: L'Antica pizzeria Da Michele - Posizione: Via Cesare Sersale, 1
3) ID: 587557654 - Nome Ristorante: 50 Kalo' - Posizione: Piazza Sannazzaro

Scegli l'id del ristorante da cui vuoi ordinare: 1
Menu' in arrivo dal ristorante 'Sorbillo'...
Menu' ricevuto, puoi effettuare l' ordine

1) Stampa Menu'
2) Aggiungi le pietanze al carrello per l'ordine inserendone l'ID
3) Completa l'ordine e invia


```

Arrivato il menù(lista di pietanze), l'utente potrà effettuare un nuovo ordine scegliendo da un apposito menù a tendina. Potrà premere 1 per visualizzare la lista delle pietanze , 2 per inserire l'ID di una nuova pietanza nel carrello e 3 per completare l'ordine. Una volta completato l'ordine, il costo totale verrà detratto dal saldo del client e verrà inviato l'ordine(lista delle pietanze scelte ,conto e l'ID ordine) e le informazioni del client (nome, cognome e ID client).



```
eduardo@eduardo-VirtualBox: ~/Documenti/Università_Parthenope/Reti di Calcolatori/Proge...
[pietanza: Pappardelle al ragu' di anatra - Costo: 12.00] Aggiunta al carrello

1) Stampa Menu'
2) Aggiungi le pietanze al carrello per l'ordine inserendone l'ID
3) Completa l'ordine e invia

2
Id pietanza: 6

[pietanza: Creme Caramel - Costo: 4.00] Aggiunta al carrello

1) Stampa Menu'
2) Aggiungi le pietanze al carrello per l'ordine inserendone l'ID
3) Completa l'ordine e invia

3
Ordine completato

Ordine creato correttamente,ecco il tuo ID ordine: [45010645] ed il tuo nuovo saldo: [3166.50]

Informazioni dell' utente inviate al server
Ordine [45010645] inviato al server, in attesa di un rider per la consegna...

█
```

Appena un rider si conatterà al ristorante a cui il client ha richiesto un nuovo ordine, il rider riceverà il client ID e l'ID ordine del client richiedente mentre il client riceverà l'ID ed il nome del rider che gli sta effettuando la consegna.

```
eduardo@eduardo-VirtualBox: ~/Documenti/Università_Parthenope/Reti di Calcolatori/Proge...
Id pietanza: 6
[pietanza: Creme Caramel - Costo: 4.00] Aggiunta al carrello
1) Stampa Menu'
2) Aggiungi le pietanze al carrello per l'ordine inserendone l'ID
3) Completa l'ordine e invia
3
Ordine completato
Ordine creato correttamente,ecco il tuo ID ordine: [45010645] ed il tuo nuovo saldo: [3166.50]
Informazioni dell' utente inviate al server
Ordine [45010645] inviato al server, in attesa di un rider per la consegna...
Il rider di ID: [471642436] e nome: [Giorgio] sta effettuando la tua consegna (ID ordine : [45010645])
presso il tuo domicilio in: [Via Luca Giordano 38]...
Il rider di ID: [471642436] e nome: [Giorgio] ha completato la tua consegna (ID ordine : [45010645]),
arrivederci
Rimanere connessi ? s/n
█
```

Completata la consegna , a client verrà chiesto se riconnettersi per effettuare un altro ordine.

Dettagli di esecuzione del rider (client)

Quando il programma rider verrà lanciato, l'utente dovrà inserire un **nome** ed un **cognome** in più verrà generato un **ID** . Successivamente il rider effettuerà una connessione con il server di instradamento e se quest'ultima avrà successo, il rider rimarrà in attesa che il server gli invii la lista di ristoranti disponibili in quel momento

```
eduardo@eduardo-VirtualBox: ~/Documenti/Università_Parthenope/Reti di Calcolatori/Progett...
[Address type]: IPv4
[Address]: 127.0.0.1

-----INFORMAZIONI RIDER-----
Inserisci il tuo nome: Giorgio

Inserisci il tuo cognome: Verdi

*****FINE INFORMAZIONI RIDER*****
*****RIDER*****
ID: 471642436
Nome: Giorgio
Cognome: Verdi
*****

server: 127.0.0.1 port: 8989
Fri Nov 5 16:28:08 2021 : Sei il [rider] numero: [2]

1) ID: 638178286 - Nome Ristorante: Sorbillo - Posizione: Via Tribunali 32
2) ID: 260865843 - Nome Ristorante: L'Antica pizzeria Da Michele - Posizione: Via Cesare Sersale, 1
3) ID: 587557654 - Nome Ristorante: 50 Kalo' - Posizione: Piazza Sannazzaro

Scegli l'id del ristorante con cui vuoi connetterti come rider: 
```

Scelto il ristorante, il rider si connetterà ad esso e verrà messo in attesa fino a quando non ci sarà un nuovo ordine. Appena ci sarà un ordine, verrà sbloccato uno dei rider connessi che confermerà di voler effettuare l'ordine e riceverà l'ordine con le pietanze, il conto ed il relativo ID e le informazioni dell'utente con il relativo ID client

```
eduardo@eduardo-VirtualBox: ~/Documenti/Università_Parthenope/Reti di Calcolatori/Progett...
Ordine ricevuto
In attesa di confermare un nuovo ordine
Ordine confermato, attendo il client ID che ha richiesto l'ordine...
Riepilogo ordine

*****UTENTE*****
ID: 837300382
Nome: Giovanni
Cognome: Rossi
Posizione: Via Luca Giordano 38
*****

*****NUOVO ORDINE*****
ID: 45010645

Pietanza: Pennette alla boscaiola - Costo: 7.50
Pietanza: Spaghetti cacio e pepe - Costo: 10.00
Pietanza: Pappardelle al ragu' di anatra - Costo: 12.00
Pietanza: Creme Caramel - Costo: 4.00

conto: 33.50
*****
```

Ricevuto l'ordine e le informazioni del client per capire dove effettuare la consegna, inizierà la consegna e quando l'avrà completata, invierà una notifica al ristorante

che a sua volta invierà al server e successivamente verrà inviata anche al client per far terminare l'ordine.

```
Consegna in corso al client id: [837300382] che risiede in [Via Luca Giordano 38]...  
Consegna effettuata al client id: [837300382] che risiede in [Via Luca Giordano 38], arrivederci  
Rimanere connessi ? s/n  
█
```

Dettagli di esecuzione del server

Il server è in attesa fin quando un client non gli richiederà un servizio.
Una volta stabilita la connessione, chiede al client quale ristorante scegliere.
In base alla scelta, si connette con l'opportuno ristorante.

```
eduardo@eduardo-VirtualBox: ~/Documenti/Università_Parthenop...  
eduardo@eduardo-VirtualBox:~/Documenti/Università_Parthenope/Reti di Calcolatori  
/Progetto_RETI_Eduardo_Autore_0124001586/Codice/server_delivery$ ./server  
Server Delivery food in attesa di connessioni...  
█
```

Scelto il ristorante e verificato che sia ancora connesso, ne appariranno le informazioni ed il server tenterà di connettersi ad esso.

```
[client] n° [2] pronto per essere servito  
  
In attesa che il client scelga un ristorante...  
-----ID RISTORANTE: 638178286-----  
Nome Ristorante: Sorbillo  
Posizione: Via Tribunali 32  
IP: 127.0.1.1  
Porta: 4000  
^  
|
```

Una volta connesso, il server richiederà al ristorante il menù che invierà successivamente al client.

Il server attenderà un ordine e le informazioni dell'utente da parte del client e appena li avrà ricevuti, li stamperà a video e inoltrerà il nuovo ordine al ristorante.

```
eduardo@eduardo-VirtualBox: ~/Documenti/Università_Parthenope/Reti di Calcolatori/Progetto_RETI...
|
Mi sto connettendo a: Sorbillo all'indirizzo 127.0.1.1 e porta 4000...
client/ristorante: 127.0.1.1 port: 4000
Fri Nov 5 16:19:33 2021 : Sei il [server] numero: [1]

In attesa che il ristorante invii il menu'...
>> Menu ricevuto da [Sorbillo]
>> Menu' inviato al client

In attesa che l' utente invii l'ordine...
>> Nuovo ordine ricevuto
-----ID ORDINE: 45010645-----
Pietanza: Pennette alla boscaiola - Costo: 7.50
Pietanza: Spaghetti cacio e pepe - Costo: 10.00
Pietanza: Pappardelle al ragu' di anatra - Costo: 12.00
Pietanza: Creme Caramel - Costo: 4.00

conto: 33.50
-----
>> Informazioni utente ricevute :
*****UTENTE*****
ID: 837300382
Nome: Giovanni
Cognome Rossi
Posizione: Via Luca Giordano 38
*****

>> Inoltro l'ordine al ristorante contattato
```

Inviato l'ordine al ristorante, attenderà fin quando non riceverà una notifica di conferma consegna con le informazioni del rider che ha confermato la consegna, in questo momento il server invierà al client le informazioni del rider con l'ID che gli sta effettuando la consegna ed al ristorante le informazioni del client con ID che ha effettuato la nuova richiesta.

Infine il server rimarrà in attesa di una notifica di consegna completata da parte del ristorante, quindi invierà la notifica di consegna completata al client per terminare l'ordine.

```
eduardo@eduardo-VirtualBox: ~/Documenti/Università_Parthenope/Reti di Calcolatori/Progetto_RETI...
Richiesta dall'host: 127.0.0.1, porta: 36720
Server Delivery food in attesa di connessioni...

-----

[rider] n° [1] pronto per essere servito
>> Lista di ristoranti inviata al rider [1]
rider 1 servito
-----

Richiesta dall'host: 127.0.0.1, porta: 36722
Server Delivery food in attesa di connessioni...

-----

[rider] n° [2] pronto per essere servito
>> Lista di ristoranti inviata al rider [2]
rider 2 servito
-----

>> Il rider con ID: [471642436] ,ricevuto dal ristorante [Sorbillo], e' pronto per la consegna
>> informazioni rider [471642436] inoltrate al client [837300382]
>> Informazioni del client di ID: [837300382] e Nome:[Giovanni ] inviate al ristorante [Sorbillo]
>> Consegna effettuata al client di ID: [837300382] e nome: [Giovanni ] all'indirizzo [Via Luca Giordano 38]
dal rider di ID: [471642436] e nome: [Giorgio]
client [2] servito
```

Oltre alla gestione del client, il server accetterà connessioni da parte dei ristoranti per registrarli nella lista dei ristoranti disponibili (da inviare al client) e da parte dei nuovi rider per comunicare loro la lista di ristoranti disponibili per la connessione.

Dettagli di esecuzione del Ristorante

Il ristorante dapprima sarà solo client e permetterà l'inserimento da input delle informazioni del ristorante (nome, posizione e menù) e la connessione al server per comunicare il proprio indirizzo e la propria porta così che il server possa inviare il ristorante al client come ristorante disponibile e quindi per permettere la connessione da parte del server ad esso in un secondo momento.

Una volta comunicati indirizzo e porta del ristorante, esso diventerà anche server e sarà pronto ad accettare nuove connessioni da parte del server o dei riders.

Appena il server si conatterà al ristorante, il ristorante invierà al server il proprio menù e rimarrà in attesa di ricevere un nuovo ordine.

Ricevuto un nuovo ordine da parte del server (che a sua volta l'ha ricevuto dal client)

Rimarrà in attesa che si colleghino nuovi rider per prendere in carico il nuovo ordine e quindi la nuova consegna.

A screenshot of a terminal window titled 'eduardo@eduardo-VirtualBox: ~/Documenti/Università_Parthenop...'. The terminal has a dark blue background with white text. The log shows the server's status and actions: it starts with the server IP and port (127.0.0.1, 8989), then a timestamp and a message about the restaurant number [1]. It then shows the restaurant [Sorbillo] waiting for connections on port [4000], followed by a request from host 127.0.0.1 on port 36970. The log continues with the restaurant waiting again, then a message '[server] n° [1] pronto per essere servito'. This is followed by a prompt '>> Menu' inviato al server', then 'In attesa di un ordine utente dal server...', and finally '>> Nuovo ordine ricevuto, invio richiesta ai riders connessi...'. A cursor is visible at the bottom left of the terminal window.

```
server: 127.0.0.1 port: 8989
Fri Nov 5 16:11:31 2021 : Sei il [ristorante] numero: [1]

Ristorante [Sorbillo] in attesa di connessioni sulla porta [4000] ...
Richiesta dall'host: 127.0.0.1, porta: 36970

Ristorante [Sorbillo] in attesa di connessioni sulla porta [4000] ...

[server] n° [1] pronto per essere servito
>> Menu' inviato al server

In attesa di un ordine utente dal server...
>> Nuovo ordine ricevuto, invio richiesta ai riders connessi...
█
```

Appena un rider si conatterà al ristorante con l'ordine in sospeso, le sue informazioni verranno stampate e prenderà in carico l'ordine inviando la conferma al ristorante che a sua volta invierà le informazioni del rider al server e riceverà le informazioni del client che verranno dapprima stampate e successivamente inoltrate al rider per permettergli di effettuare la consegna.

Una volta terminata la consegna da parte del rider, esso invierà al ristorante una notifica di consegna avvenuta che verrà successivamente inoltrata al server ed infine al client per far terminare l'ordine.


```
eduardo@eduardo-VirtualBox: ~/Documenti/Università_Parthenope/Reti di Calcolatori/Progetto_RETI_Eduardo_Au...
[rider] n° [1] pronto per essere servito

*****RIDER*****
ID: 471642436
Nome: Giorgio
Cognome: Verdi
*****

In attesa di una nuova richiesta...
>> Il rider Giorgio effettuerà la consegna
>> Nuovo ordine disponibile per il rider [Giorgio]
>> ID rider : [471642436] inviato al server
>> Conferma effettuata dal rider [Giorgio]

In attesa che il server invii l'id del client al rider...
*****UTENTE*****
ID: 837300382
Nome: Giovanni
Cognome: Rossi
Posizione: Via Luca Giordano 38
*****

>> Il client [837300382] sta per ricevere la sua ordinazione dal rider di ID: [471642436] a [Via Luca Giordano 38]

>> Il rider [Giorgio] ha ricevuto il client id [837300382]
>> Consegna effettuata al client id: [837300382] che risiede in [Via Luca Giordano 38], arrivederci

server 1 servito
```

Manuale dell'utente

Per avviare correttamente il programma è necessario eseguire prima di tutto il server. Successivamente è necessario avviare uno o più ristoranti, inserire le informazioni di questi ultimi e lasciarli in attesa.

Adesso possiamo avviare uno o più client, inserire le informazioni di questi ultimi e scegliere uno degli N ristoranti che appariranno nella schermata. Scelto un ristorante, ci comparirà una menù di scelta dove potremo premere 1 per stampare il menù ricevuto dal ristorante, 2 per aggiungere una delle pietanze al carrello e 3 per completare l'ordine.

Effettuato l'ordine, non dovremo fare altro che attendere che torni disponibile qualche rider per la consegna.

Quindi eseguiamo uno o più riders, inseriamo le informazioni di questi ultimi e attendiamo che diano conferma al ristorante, nel mentre il client si sarà sbloccato poiché gli sarà arrivata una notifica dal server di ordine in consegna ed appena il rider avrà completato la consegna, al client arriverà un'ultima modifica di consegna completata dal suddetto rider. Tutti i riders connessi che non hanno ancora una richiesta da eseguire, rimarranno in attesa fin quando non verranno fornite altre richieste quindi è possibile avviare i riders anche prima che il client effettui un nuovo ordine.

Istruzioni per la compilazione

- Per compilare il server bisogna andare nella cartella “**Server_Delivery**”, aprire il terminale e digitare il seguente comando:
`gcc server.c -o server -lpthread queue.c llist.c wrapperSysCall.c abstract_user.c`
oppure digitare il comando “make” così che il makefile esegua automaticamente il comando sopra citato.
- Per compilare il ristorante bisogna andare nella cartella “**ristorante**”, aprire il terminale e digitare il seguente comando:
`gcc ristorante.c -o ristorante -lpthread queue.c llist.c wrapperSysCall.c abstract_user.c`
oppure digitare il comando “make” così che il makefile esegua automaticamente il comando sopra citato.
- Per compilare il client bisogna andare nella cartella “**client**”, aprire il terminale e digitare il seguente comando:
`gcc client.c -o client queue.c llist.c wrapperSysCall.c abstract_user.c`
oppure digitare il comando “make” così che il makefile esegua automaticamente il comando sopra citato.
- Per compilare il rider bisogna andare nella cartella “**rider**”, aprire il terminale e digitare il seguente comando:
`gcc rider.c -o rider queue.c llist.c wrapperSysCall.c abstract_user.c`
oppure digitare il comando “make” così che il makefile esegua automaticamente il comando sopra citato.

Istruzioni per l'esecuzione

- Per eseguire il server bisogna andare nella cartella “**Server_Delivery**”, aprire il terminale e digitare il seguente comando:
`./server`
- Per eseguire il ristorante bisogna andare nella cartella “**ristorante**”, aprire il terminale e digitare il seguente comando:
`./ristorante`
- Per eseguire il client bisogna andare nella cartella “**client**”, aprire il terminale e digitare il seguente comando:

`“./client”`

- Per eseguire il rider bisogna andare nella cartella “`rider`”, aprire il terminale e digitare il seguente comando:
`“./rider”`