Università degli Studi di Napoli "Parthenope"

Progetto Di Reti Di Calcolatori

Marketplace

Pierluigi Scotto 0124001277
Gennaro Traino 0124001105

Sommario

Traccia

Si vuole realizzare un sistema per la gestione di negozi virtuali costituito dalle seguenti entità:

- **ServerM:** mantiene la lista dei negozi virtuali e dei prodotti di ogni negozio virtuale. Interagisce con ServerN e ServerC.
- **ServerN:** consente ai ClientN di operare sul ServerM. In particolare consente di creare un nuovo negozio virtuale, eliminare un negozio virtuale ed aggiungere ed eliminare prodotti da un negozio virtuale.
- **ServerC:** consente ai Client di operare sul ServerM. In particolare consente di ricevere l'elenco dei negozi virtuali, ricevere l'elenco dei prodotti di un negozio virtuale e ricercare un prodotto in un negozio virtuale.
- **ClientN:** consente al negoziante di gestire i propri negozi virtuali (ed i relativi prodotti) memorizzati sul ServerM, usando il ServerN come tramite. Ogni negoziante pu`o gestire più negozi virtuali.
- **Client:** consente all'utente di interagire con i negozi virtuali memorizzati sul ServerM usando il ServerC come tramite. In particolare, consente all'utente di inserire i prodotti contenuti in diversi negozi virtuali in una lista di acquisti e di visualizzare la lista di acquisti.

Descrizione del Progetto

Descrizione Generale

Lo sviluppo dell'applicazione prevede 5 tipi di entità diverse:

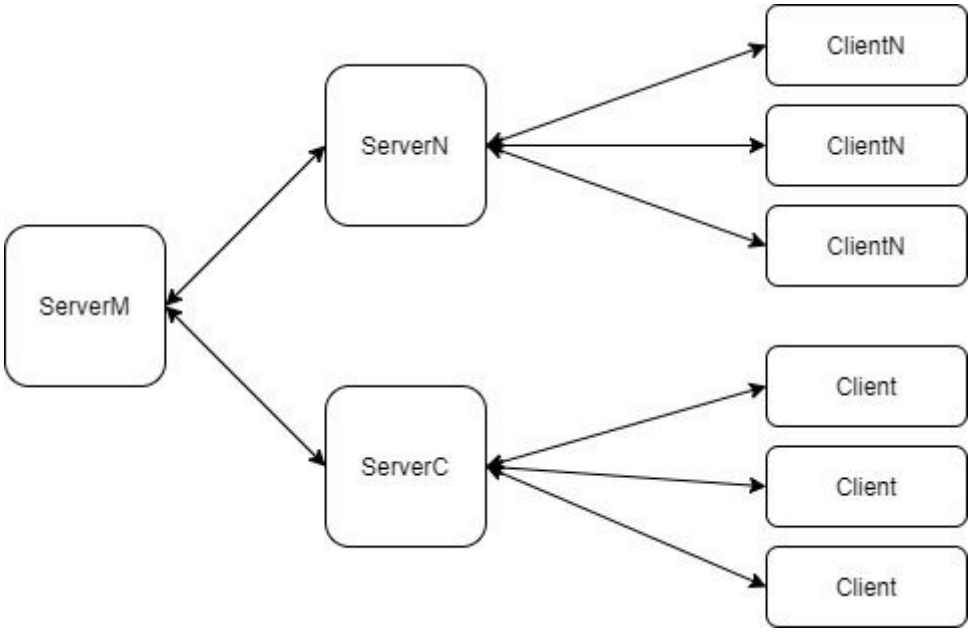
- **ServerM**, è il server principale. E' stato sviluppato come **server iterativo**, su cui si mantengono la **lista dei negozi con i relativi proprietari ed i relativi prodotti**. In aggiunta esegue le varie operazioni che sono inviate dal ServerC e ServerN.
- **ClientN,** rappresentano i proprietari di uno o più negozi, i quali, dopo essersi collegati con **username** e **password**, scelgono tramite una semplice interfaccia le operazioni da effettuare sui propri negozi: come aggiunta o rimozione di un prodotto o di un nuovo negozio virtuale. Essi inoltreranno le richiesta al serverN.
- **ServerN**, è il server che accoglie le richieste dei clientN, è stato sviluppato come un **server concorrente**. Mantiene una **lista di utenti** e interagisce con il ServerM per le operazioni richieste dai ClientN. Ha anche il compito di verificare se l'utente che vuole operare su un particolare negozio sia il proprietario di esso e quindi autorizzare l'operazione.
- **Client**, rappresentano gli utenti finali. Tramite un'interfaccia sceglie di visualizzare una lista di prodotti di un determinato negozio e di inserirlo nel carrello (lista che manterrà localmente). Può visualizzare i suoi prodotti inseriti nel carrello.
- **ServerC**, è il server che supporta le operazioni dei Client. In particolare, dopo una richiesta al ServerM, mostrerà ad essi i cataloghi dei prodotti dei negozi sempre aggiornati ed inoltre permette di ricercare un particolare prodotto all'interno della lista per aggiungerlo al carrello.

Scelte implementative

Per lo sviluppo del programma è stato usato il linguaggio C ed è stato sviluppato su piattaforma UNIX utilizzando i socket per la comunicazione tra processi.
Le socket utilizzate nel programma sono di tipo **`SOCK_STREAM`** (streaming socket).
La scelta è stata dettata dall'esigenza di ricevere e inviare pacchetti in modo affidabile focalizzandosi sull'affidabilità piuttosto che sulla velocità (per questo motivo si è evitato il protocollo UDP).
E' stata utilizzata la macro **`AF_INET`** che rappresenta la famiglia di protocolli IPv4, che, insieme alla **`SOCK_STREAM`**, determina una connessione TCP/IP.

Schemi e Architettura Applicazione

Di seguito viene mostrato un piccolo schema che disegna un quadro generale sull'architettura del programma:



Nella figura sono presenti le 5 entità presentate in precedenza e le frecce rappresentano le loro interazioni possibili.

ServerM

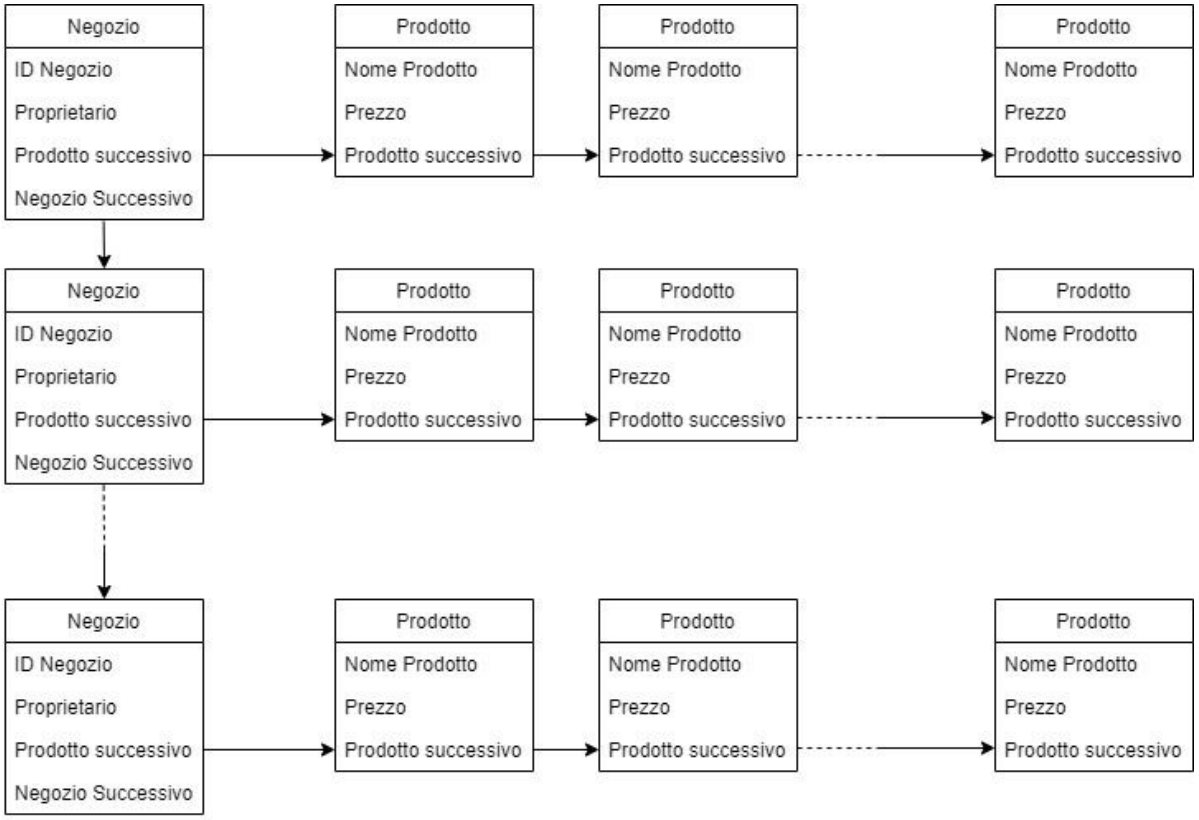
Il ServerM è sviluppato come server iterativo che accetta le richieste di connessione da parte del **ServerN** e del **ServerC**.
Mantiene inoltre una lista dei negozi con i relativi prodotti.
La lista dei negozi è formata dai seguenti campi:

- l'id del negozio
- il nome del proprietario
- un puntatore che punta al prossimo negozio
- un puntatore che punta al prossimo prodotto

Mentre la lista dei prodotti è formata dai seguenti campi:

- il nome prodotto
- il prezzo prodotto
- un puntatore che punta al prossimo prodotto

Di seguito viene riportata un'immagine di com'è strutturata la linkedlist:

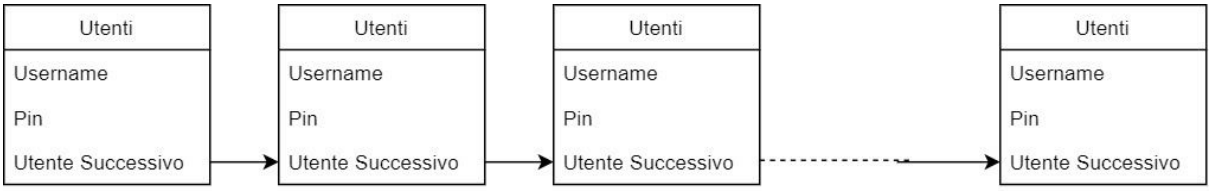


Il server quindi alloca la lista di negozi e prodotti e la gestisce con le operazioni possibili a seconda delle richieste derivanti dai due server C ed N.
In particolare il serverM sarà in attesa di una connessione da parte degli altri due server. Una volta stabilita la connessione aspetta la ricezione di un codice che specifica l'operazione che il server collegato vuole eseguire.
Per ogni operazione, vi è un procedimento diverso con cui i due server si scambiano dati che rendono possibile l'esecuzione di tali richieste e dati per la risposta per il richiedente.
In particolare le operazioni che il serverM può effettuare sono:

- Riceve richiesta client
- Inviare la lista dei prodotti di tutti i negozi ai due server sempre aggiornata.
- Inserire un nuovo prodotto ricevuto dal ServerN nella lista.
- Eliminare un prodotto ricevuto dal ServerN dalla lista dei prodotti.
- Inserire un nuovo negozio ricevuto dal ServerN nella lista dei negozi.
- Eliminare un negozio ricevuto dal ServerN dalla lista dei negozi.
- Invia lista dei negozi di uno specifico utente al ServerN.
- Controllare se un utente è il proprietario di un negozio e inviare la risposta al Server richiedente.

ServerN

Il serverN è stato sviluppato come un **server concorrente** che accetta le connessioni da parte del **clientN** e gestisce le richieste creando ad ogni connessione un figlio che gestisce il collegamento con il client. Mantiene una lista di utenti che possiedono uno o più negozi.
Di seguito viene mostrata un'immagine di come è strutturata la lista degli utenti:



Ad ogni connessione dei ClientN, il ServerN richiede le credenziali per identificare l'utente che si sta connettendo.
Dopo che il ClientN si è collegato utilizzando le sue credenziali, il ServerN si collega al ServerM per richiedere la lista aggiornata dei negozi e dei prodotti da inviare al ClientN.
Il compito del ServerN è quello di ricevere le richieste di operazioni da parte dei ClientN ed inviarle al ServerM per poterle eseguire. Dopo che sono state eseguite le operazioni, riceve la risposta da parte del ServerM, il quale inoltra il risultato o la risposta al ClientN che ha effettuato la richiesta.
Le richieste possibili sono le seguenti:

- Inserisci un prodotto in un negozio, ricevendo nome e prezzo del prodotto e l'id del negozio da parte del clientN, inviando tali dati al ServerM con il relativo codice dell'operazione.
- Eliminare un prodotto, ricevendo nome prodotto e proprietario dal ClientN e li manda insieme al codice operativo al ServerM per eseguire l'operazione.
- Creare un nuovo negozio, inviando il nome del proprietario ricevuto dal ClientN al ServerM con il codice operazione.
- Eliminare un negozio, inviando l'id negozio ricevuto al ServerM insieme al codice operazione.
- Modificare il negozio su cui operare, inviando i soliti dati utente al ServerM.
- Mostrare negozi al client collegato, inviando dati utente e ricevendo la lista negozi dal ServerM, e poi inoltrarla al ClientN.

ServerC

Il ServerC è stato sviluppato come **server concorrente** che accetta le connessioni da parte del Client e gestisce le richieste creando ad ogni connessione un figlio che gestisce il collegamento con il Client.
Ad ogni connessione dei Client, il ServerC si connette al ServerM per ricevere la lista aggiornata dei negozi e dei prodotti.
Il suo compito è quello di ricevere le richieste di operazioni ricevute dal Client, elaborare l'operazione e infine inviare il risultato al Client.
Le richieste possibili sono le seguenti:

- Invia la lista dei prodotti di un particolare negozio al Client. Sfruttando la lista aggiornata che richiede ad ogni nuova operazione al ServerM.

- Riceve il nome di un prodotto e lo ricerca nella lista, inviando al client il prodotto cercato se presente.

ClientN

Il ClientN rappresenta il negoziante che gestisce i suoi negozi virtuali.
Una volta eseguito il ClientN il negoziante dovrà loggarsi inserendo le proprie credenziali ed inviarle al ServerN, se passerà il controllo potrà scegliere il negozio su cui operare a patto che sia di sua proprietà.
Una volta scelto il negozio su cui operare potrà quindi usufruire delle varie operazioni messe a disposizione:

- Uscire dal programma
- Inserire il prodotto in un negozio, inviando nome e prezzo al ServerN e il relativo codice dell'operazione.
- Elimina il prodotto da un negozio, inviando il nome del prodotto,l'id del negozio e il codice operazione, ricevendo una conferma dal ServerN.
- Creare nuovo negozio virtuale, inviando username al ServerN e ricevendo un check di conferma.
- Eliminare un negozio virtuale, inviando username al ServerN e ricevendo un check di conferma.
- Cambiare il negozio su cui operare, inviando al ServerN il nome e l'id del negozio, ricevendo una conferma.

Client

Il client rappresenta l'utente finale che gestisce una lista di acquisti. Dapprima il Client si collega al ServerC e riceve la lista di negozi con cui poi scegliere i vari prodotti. Tramite un'interfaccia può scegliere varie operazioni:

- Uscire dal programma
- Inserire un prodotto nel carrello, inviando l'id del negozio e il nome del prodotto di cui si vuole acquistare, infine riceve il prezzo dal ServerC e lo inserisce nel carrello.
- Stampa la lista di acquisti.
- Visualizzare lista prodotti di un negozio, inviando l'id del negozio al ServerC, ricevendo il consenso se esiste il negozio di cui si sta cercando, e se esiste riceve la lista dei prodotti di quel negozio.

Dettagli implementativi

Strutture Dati (List.h)

Sono state utilizzate 3 struct per mantenere le 3 linked lists: una per la lista dei negozi, una per quelle dei prodotti che come spiegato in precedenza sono collegate ad un nodo della precedente, infine quella per gli utenti.

```
struct Negozio {
  int id;
  struct Negozio *next;
  struct Prodotto *pnext;
  char *proprietario;
};
struct Prodotto {
  char *nome;
  float prezzo;
  struct Prodotto *pnext;
};
struct Utente {
  char *username;
  char *pin;
  struct Utente *unext;
};
```

Le funzioni per la operazioni di gestione su queste linked list, come aggiunta o rimozione di nodi, sono contenute all'interno della libreria **`List.h`**

```
struct Negozio *creaListaNegozio(struct Negozio *testa, int id);
struct Utente *creaListaUtenti(struct Utente *testa, char *nome, char *pin);
struct Prodotto *creaListaProdotti(char *nome, float prezzo,
                                   struct Prodotto *testa);
struct Negozio *aggiungiNegozio(struct Negozio *testa, int id);
struct Utente *aggiungiUtente(struct Utente *testa, char *username, char *pin);
void *aggiungiProdotto(char *nome, float prezzo, struct Negozio *testa, int id);
struct Prodotto *aggiungiProdottoLista(struct Prodotto *testa, char *nome,
                                       float prezzo);
struct Negozio *eliminaNegozio(struct Negozio *negozio, int id);
struct Prodotto *cercaProdotto(struct Negozio *negozio, int id, char *nome);
struct Prodotto *eliminaProdotto(struct Negozio *negozio, int id, char *nome);
void stampaLista(struct Negozio *negozio);
void stampaProdotti(struct Prodotto *prodotti);
void stampaUtenti(struct Utente *utenti);
void assegnaProprietario(char *proprietario, int id_negozio,
                         struct Negozio *lista);
```

```c
//Controlla se un utente è nella lista
int checkUtente(char *nome, char *pin, struct Utente *lista);
//Controlla se un utente possiede il negozio passato come parametro
int checkProprietario(char *nome, struct Negozio *lista, int id);
```

Libreria gestione errori (Wrapped.h)

Libreria che contiene un'estensione delle funzioni socket e sulle funzioni di IO, come send e recv, per il controllo degli errori.
Contiene inoltre funzioni per lo scambio di campi e variabili su socket, con controllo degli errori.

```c
int Socket(int, int, int);
void Connect(int, struct sockaddr *, socklen_t);
void Bind(int, struct sockaddr *, socklen_t);
void Listen(int, int);
int Accept(int, struct sockaddr *, socklen_t *);
void Inet_pton(int, const char *, void *);
// Ricezione
void riceviCheck(int socket, int *check);
void riceviProdotto(int socket, char *nome, float *prezzo, int *id);
void riceviIdProdotto(int socket, char *nome, int *id);
void riceviUtente(int socket, char *nome, char *pin);
void riceviProprietario(int socket, char *nome, int *id);
void riceviPrezzo(int socket, float *prezzo);
void riceviIdNegozio(int socket, int *id);
void riceviUsername(int socket, char *username);
// Invio
void inviaCheck(int socket, int *check);
void inviaProdotto(int socket, char *nome, float *prezzo, int *id);
void inviaIdProdotto(int socket, char *nome, int *id);
void inviaProprietario(int socket, char *nome, int *id);
void inviaScelta(int socket, int *scelta);
void inviaUtente(int socket, char *nome, char *pin);
void inviaPrezzo(int socket, float *prezzo);
void inviaIdNegozio(int socket, int *id);
void inviaUsername(int socket, char *username);
```

Libreria scambio strutture dati (IO.h)

Libreria che contiene tutti i prototipi delle funzioni che permettono lo scambio di strutture dati tra client e Server.

```c
/* SCAMBIO DI LISTA NEGOZI E PRODOTTI TRA SERVER */
void inviaListaProdotti(int socket, struct Negozio *lista);
void invioListaNegozi(int socket, struct Negozio *lista);
void inviaListaNegoziProprietario(int socket, struct Negozio *lista,
```

```
                                    char *proprietario);
struct Negozio *ricezioneListaNegozi(int socket, struct Negozio *testa);
struct Negozio *ricezioneListaProdotti(int socket, struct Negozio *lista);
/* SCAMBIO LISTA DI SOLI PRODOTTI DA MOSTRARE AL CLIENT */
void *inviaprodotti_di(int socket, struct Negozio *lista, int id);
struct Prodotto *riceviprodotti_di(int socket, struct Prodotto *prodotti);
```

invioListaNegozi():

Per inviare una listed link, si deve necessariamente inviare campo per campo ogni nodo, lasciando al ricevente il compito di ricostruire la lista fino alla ricezione dell'id = -1 che vale come "protocollo" per fermare l'invio e la ricezione.

```
/**Invia l'id della lista dei negozi al ServerC o ServerN
 * - socket di connessione
 * - lista negozi
 **/
void invioListaNegozi(int socket, struct Negozio *lista) {
  // Scorri lista
  while (lista) {
    int id = lista->id;
    inviaIdNegozio(socket, &id);
    lista = lista->next;
  }
  // Condizione per fermare le recv del ServerC / ServerN
  int id = -1;
  inviaIdNegozio(socket, &id);
}
```

inviaListaNegoziProprietario():

Con lo stesso principio della precedente funzione, invierà solamente i negozi in cui il proprietario corrisponde a quello passato come argomento.

```
/**Invia l'id della lista di negozi con i proprietari al ClientN
 * - socket di connessione
 * - lista negozi
 * - proprietario negozio
 **/
void inviaListaNegoziProprietario(int socket, struct Negozio *lista,
                                  char *proprietario) {
  // Scorri lista
  while (lista) {
    int id = lista->id;
    // Controlla proprietario
    if (strcmp(lista->proprietario, proprietario) == 0 ||
        proprietario == NULL) {
```

```
      inviaIdNegozio(socket, &id);
    }
    lista = lista->next;
  }
```

ricezioneListaNegozi():

Si riceve campo per campo ogni nodo della lista e si ricostruisce con le apposite funzioni scritte in **`list.c`**. Alla ricezione di `id = -1` cesserà di ricevere campi.

```
/** Ricezione id lista di negozi dal ServerM
 * - socket di connessione
 * - lista negozi
 **/
struct Negozio *ricezioneListaNegozi(int socket, struct Negozio *lista) {
  int id = 0; // Id negozio
  // Crea testa lista negozi
  riceviIdNegozio(socket, &id);
  lista = creaListaNegozio(lista, id);
  // Inserisce negozio
  while (1) {
    riceviIdNegozio(socket, &id);
    // Condizione per fermare la ricezione di negozi
    if (id == -1)
      break;
    // Riceve id e costruisce lista
    lista = aggiungiNegozio(lista, id);
  }
  return lista;
}
```

inviaListaProdotti():

Funzione che invia ogni nodo campo per campo, e invia `id` e `prezzo = -1` alla fine.

```
/** Invia la lista dei prodotti di tutti i negozi al ServerC o ServerN
 * - socket di connessione
 * - lista negozi
 **/
void inviaListaProdotti(int socket, struct Negozio *lista) {
  struct Prodotto *prodotto;
  // Scorre lista negozi
  while (lista) {
    prodotto = lista->pnext;
    // Scorre lista prodotti
    // Invio prodotti al ServerC/ServerN
    while (prodotto) {
```

```
      char nome[25]; // Nome prodotto
      strcpy(nome, prodotto->nome);
      float prezzo = prodotto->prezzo; // Prezzo prodotto
      int id = lista->id;              // Id negozio
      // Invio prodotto
      inviaProdotto(socket, nome, &prezzo, &id);
      prodotto = prodotto->pnext;
    }
    lista = lista->next;
  }
  // Condizioni per terminare
  char fine[25] = "fine";
  float prezzo = -1;
  int id = -1;
  inviaProdotto(socket, fine, &prezzo, &id);
}
```

ricezioneListaProdotti():

Si riceve campo per campo ogni nodo della lista prodotti (nome,prezzo ed id del negozio) e si ricostruisce con le apposite funzioni scritte in **`list.c`**. Alla ricezione di `id = -1` cesserà di ricevere campi.

```
/* Riceve la lista dei prodotti di tutti i negozi dal ServerM
 * - socket di connessione
 * - lista negozi
 **/
struct Negozio *ricezioneListaProdotti(int socket, struct Negozio *lista) {
  struct Prodotto *prodotto = NULL; // Testa lista prodotto
  char nome[25];                    // Nome prodotto
  float prezzo;                     // Prezzo prodotto
  int id;                           // Id negozio
  // Riceve nome,prezzo e id
  while (1) {
    riceviProdotto(socket, nome, &prezzo, &id);
    // Condizioni per terminare
    if (prezzo == -1 && id == -1)
      return lista;
    // Riceve il prodotto e crea lista
    prodotto = aggiungiProdotto(nome, prezzo, lista, id);
  }
  return lista;
}
```

inviaprodotti_di():

Invia i prodotti del negozio che corrisponde all'id passato come argomento.

```c
/* Invia prodotti di un negozio al Client
 * - socket di connessione
 * - lista negozi
 * - id negozio
 **/
void *inviaprodotti_di(int socket, struct Negozio *lista, int id) {

  // Scorre lista
  while (lista) {
    struct Prodotto *prodotto = NULL; // Testa lista prodotto
    char nome[25];                    // Nome prodotto
    float prezzo;                     // Prezzo prodotto
    // Se il prodotto è in testa
    if (lista->id == id) {
      prodotto = lista->pnext;
      // Scorre la lista dei prodotti
      while (prodotto) {
        strcpy(nome, prodotto->nome);
        prezzo = prodotto->prezzo;
        // Invio nome
        if (send(socket, &nome, 25 * sizeof(char), 0) < 0) {
          perror("Errore in scrittura del nome del prodotto");
          exit(0);
        }
        // Invio prezzo
        inviaPrezzo(socket, &prezzo);
        prodotto = prodotto->pnext;
      }
      // Condizioni per terminare
      strcpy(nome, "fine");
      prezzo = -1;
      if (send(socket, &nome, 25 * sizeof(char), 0) < 0) {
        perror("Errore in scrittura del nome del prodotto");
        exit(0);
      }
      inviaPrezzo(socket, &prezzo);
    }
    lista = lista->next;
  }
}
```

riceviprodotti_di():

Riceve i prodotti del negozio che corrisponde all'id che è stato inviato nella funzione precedente.

```
/* Riceve prodotti di un negozio al Client
 * - socket di connessione
 * - lista prodotti
 **/
struct Prodotto *riceviprodotti_di(int socket, struct Prodotto *prodotti) {
  struct Prodotto *prodotto = NULL; // Testa prodotto lista
  char nome[25];                    // Nome prodotto
  float prezzo;                     // Prezzo prodotto
  // Riceve nome e prezzo di un prodotto
  // Per la testa
  if (recv(socket, &nome, 25 * sizeof(char), 0) > 0 &&
      recv(socket, &prezzo, sizeof(prezzo), 0) > 0) {
    if (prezzo == -2)
      return prodotto;
    else if (prezzo == -1)
      return prodotto;
    else
      prodotto = creaListaProdotti(nome, prezzo, prodotto);
  }
  // Per il resto
  struct Prodotto *ptemp = prodotto; // Copia testa prodotto lista
  while (recv(socket, &nome, 25 * sizeof(char), 0) > 0 &&
         recv(socket, &prezzo, sizeof(prezzo), 0) > 0) {
    if (prezzo == -1)
      break;
    else if (prezzo == -2)
      return prodotto;
    else
      ptemp = aggiungiProdottoLista(prodotto, nome, prezzo);
  }
  return prodotto;
}
```

ServerM

ServerM iterativo, definito sulla porta 1024, ad ogni connessione aspetta di ricevere tramite la funzione riceviCheck(), il comando dell'operazione che il serverN o serverC richiedono. Dopo aver svolto l'operazione chiude la connessione e ne aspetta una nuova.

```
  #define PORTA 1024
  // Creazione socket per ServerC / ServerN
  ascolto_fd = Socket(AF_INET, SOCK_STREAM, 0);
```

```
  // Assegnazione famiglia, indirizzi, porta
  server.sin_family = AF_INET;
  server.sin_addr.s_addr = htonl(INADDR_ANY);
  server.sin_port = htons(PORTA);
  // Collegamento socket alla porta 1024
  Bind(ascolto_fd, (struct sockaddr *)&server, sizeof(server));
  // Stabilisce il numero massimo di connessioni client
  Listen(ascolto_fd, 100);
  printf("\nServerM in ascolto...\n");
  while (1) {
  int check, server;
  lunghezza = sizeof(client);
  // Accetta connessioni dal ServerC / ServerN
  connessione_fd = Accept(ascolto_fd, (struct sockaddr *)&client,
&lunghezza);
  printf("\nRicezione connessione con il server");
  // Ricevi numero dal Server N / Server C
  riceviCheck(connessione_fd, &server);
  // Aggiornamento lista ServerC
  if (server == 0) {...}
  // Inserire prodotto in lista ServerN
  if (server == 1) {...}
  // Elimina prodotto dalla lista ServerN
  if (server == 2) {...}
  // Inserisce un negozio ServerC
  if (server == 3) {...}
  // Elimina un negozio ServerC
  if (server == 4) {...}
  // Controllo proprietario del negozio
  if (server == 5) {...}
  // Invio lista negozi gestiti dall'utente collegato
  if (server == 6) {...}
  // Controlla negozio utente ServerN
  else if (server == 10) {...}
  printf("\nDisconnessione con il server...");
  close(connessione_fd);
 }
```

ServerC

ServerC, definito sulla porta 1025, è sviluppato come server concorrente, ad ogni connessione del client esegue una fork e lascia gestire al figlio le operazioni richieste dal client.

Riceve il codice operazione dal client, stabilisce una connessione con il serverM sulla porta 1024 per ricevere la lista aggiornata, svolge l'operazione e chiude la connessione.

```c
#define PORTAM 1024 // Porta per il serverM
#define PORTA 1025  // Porta per il client
  /**Connessione con il ServerM**/
  int socketM;                    // Socket connessione ServerM
  struct sockaddr_in indirizzoM; // Indirizzo ServerM
  struct Negozio *lista = NULL;  // Lista negozi
  // Assegnazione indirizzo
  indirizzoM.sin_family = AF_INET;
  indirizzoM.sin_port = htons(PORTAM);

  /**Connessione con il Client**/
  int ascolto_fd;             // Socket Client per l'ascolto
  int connessione_fd;         // Socket Client per la connessione
  struct sockaddr_in server; // Indirizzo
  struct sockaddr_in client; // Indirizzo client
  socklen_t lunghezza;        // Lunghezza indirizzo client
  pid_t pid;                  // Process id
  // Creazione socket ascolto Client
  ascolto_fd = Socket(AF_INET, SOCK_STREAM, 0);
  // Assegnazione famiglia, indirizzi, porta
  server.sin_family = AF_INET;
  server.sin_addr.s_addr = htonl(INADDR_ANY);
  server.sin_port = htons(PORTA);
  // Collegamento socket client alla porta 1025
  Bind(ascolto_fd, (struct sockaddr *)&server, sizeof(server));
  // Stabilisce il numero massimo di connessioni client
  Listen(ascolto_fd, 100);
  printf("\nServerC in ascolto...\n");
  while (1) {
    lunghezza = sizeof(client);
    // Accetta connessioni dal Client
    connessione_fd = Accept(ascolto_fd, (struct sockaddr *)&client, &lunghezza);
    printf("\nRicezione connessione con il client...");

    /**Connessione al serverM
     * Per avere la lista aggiornata
     **/
    socketM = Socket(AF_INET, SOCK_STREAM, 0); // Crea socket serverM
    // Connessione al serverM
    Connect(socketM, (struct sockaddr *)&indirizzoM, sizeof(indirizzoM));
    printf("\n\tConnessione con il server...");
    // Richiede al serverM la lista aggiornata
    int campo = 0;
    inviaCheck(socketM, &campo);
```

```
    printf("\n\tRicezione lista negozi e prodotti");
    lista = NULL;
    lista = ricezioneListaNegozi(socketM, lista);
    lista = ricezioneListaProdotti(socketM, lista);
    printf("\n\tDisconnessione con il server...");
    close(socketM);
    // Creazione figlio
    if ((pid = fork()) < 0) {...}
    // Figlio
    if (pid == 0) {
      int scelta = 0; // Scelta client
      // Chiude ascolto
      close(ascolto_fd);
      printf("\n\tInvio lista al client...");
      invioListaNegozi(connessione_fd, lista); // Invia lista al client
      while (1) {
        // Riceve scelta dal client
        riceviCheck(connessione_fd, &scelta);
        /**Connessione con il server M
         * per avere la lista aggiornata**/
        // Crea e connetti al serverM
        socketM = Socket(AF_INET, SOCK_STREAM, 0);
        Connect(socketM, (struct sockaddr *)&indirizzoM, sizeof(indirizzoM));
        printf("\n\tConnessione con il server...");
        // Richiede ad M la lista aggiornata
        int campo = 0;
        inviaCheck(socketM, &campo);
        printf("\n\tRicezione lista aggiornata");
        lista = ricezioneListaNegozi(socketM, lista);
        lista = ricezioneListaProdotti(socketM, lista);
        /**Inserimento Client**/
        // Disconnessione client
        if (scelta == 0) {...}
        // Ricezione e invio prodotto
        if (scelta == 1) {...}
        // Invia lista prodotti di un negozio
        if (scelta == 3) {...}
        printf("\n\tDisconnessione con il server...");
        close(socketM);
      }
      printf("\nDisconnessione con il client...");
      // Chiudi connessione con il client
      close(connessione_fd);
      exit(0);
    } else { // Padre
      close(connessione_fd);
```

```
    }
  }
  exit(0);
}
```

ServerN

Server concorrente, definito sulla porta 1026, ad ogni connessione di un clientN, esegue una fork e lascia il figlio a gestire la connessione del clientN connesso.
Ad ogni connessione di un clientN richiede e verifica, con l'aiuto del ServerM, le credenziali e su quale negozio il client ha intenzione di operare.
Passata la prima fase, entra in un while in cui aspetta che il client invii il codice operazione, per ogni codice inviato, provvede a connettersi al serverM sulla porta 1024, concludere le operazioni ed inviare un check di completamento al clientN.

```
#define PORTAM 1024
#define PORTA 1026

int main(int argc, char const *argv[]) {
  /*****Connessione con il ServerM*****/
  int socketM;                    // Socket ServerM
  struct sockaddr_in indirizzoM; // Indirizzo ServerM
  // Assegnazione indirizzo
  indirizzoM.sin_family = AF_INET;
  indirizzoM.sin_port = htons(PORTAM);
  /*****Connessione con il Client*****/
  int ascolto_fd;             // Socket ascolto Client
  int connessione_fd;         // Socket connessione Client
  struct sockaddr_in server; // Indrizzo server
  struct sockaddr_in client; // Indrizzo client
  socklen_t lunghezza;        // Lunghezza indirizzo client
  pid_t pid;                  // Process id
  // Creazione socket Client
  ascolto_fd = Socket(AF_INET, SOCK_STREAM, 0);
  // Assegnazione famiglia, indirizzi, porta
  server.sin_family = AF_INET;
  server.sin_addr.s_addr = htonl(INADDR_ANY);
  server.sin_port = htons(PORTA);
  // Collegamento socket client alla porta 1025
  Bind(ascolto_fd, (struct sockaddr *)&server, sizeof(server));
  // Stabilisce il numero massimo di connessioni client
  Listen(ascolto_fd, 100);
  printf("\nServerN in ascolto...\n");
  while (1) {
    lunghezza = sizeof(client);
    // Accetta connessioni dal Client
    connessione_fd = Accept(ascolto_fd, (struct sockaddr *)&client, &lunghezza);
```

```
printf("\n\tRicezione connessione con client...");
// Crea figlio per assegnargli la gestione del client
if ((pid = fork()) < 0) {...}
// Figlio
if (pid == 0) {
  close(ascolto_fd); // Chiudi ascolto
  char nome[25];     // Username utente
  char pin[5];       // Pin utente
  /** Verifica se l'utente è in lista **/
  printf("\n\tRicezione utente");
  // Ricevi Username e Pin dal client
  riceviUtente(connessione_fd, nome, pin);
  // Verifica login
     ...
  }

  /** Verifica negozio proprietario **/
  int id_negozio;
  while (1) {
    printf("\n\tRicezione id negozio");
    riceviProprietario(connessione_fd, nome, &id_negozio);
    printf("\n\tConnessione con il server...");
    /**Connessione con il server M
     * per controllare se l'utente
     * proprietario del negozio scelto*/
    // Connessione con il serverM
    socketM = Socket(AF_INET, SOCK_STREAM, 0);
    Connect(socketM, (struct sockaddr *)&indirizzoM, sizeof(indirizzoM));
    // Codice che aiuta il serverM per capire l'operazione da fare
    int server = 10;
    // Manda al serverM il nome del proprietario e id negozio
    inviaCheck(socketM, &server);
    inviaProprietario(socketM, nome, &id_negozio);
    // Ricevi da serverM il consenso con 1
    riceviCheck(socketM, &check);
    inviaCheck(connessione_fd, &check);
    if (check)
      break;
  }
  printf("\n\tDisconnessione con il server");
  close(socketM);
  /** Scelta operazione del client **/
  while (1) {
    int scelta;
    riceviCheck(connessione_fd, &scelta);
    printf("\n\tRicezione operazione del client");
```

```
      // Disconnessione client
      if (scelta == 0) {...}
      printf("\n\tConnessione con il server...");
      /** Connessione con il serverM**/
      socketM = Socket(AF_INET, SOCK_STREAM, 0);
      Connect(socketM, (struct sockaddr *)&indirizzoM, sizeof(indirizzoM));
      /***Inserimento Client***/
      // Inserisce Prodotto in lista
      if (scelta == 1) {...}
      // Elimina Prodotto in lista
      if (scelta == 2) {...}
      // Inserisce negozio
      if (scelta == 3) {...}
      // Elimina negozio
      if (scelta == 4) {...}
      // Gestione altro negozio
      if (scelta == 5) {...}
      // Mostra Negozi
      if (scelta == 6) {...}
      close(socketM);
    }
    printf("\n\tDisconnessione server e client...\n");
    close(socketM);
    close(connessione_fd);
    exit(0);
  }
  else { // PADRE
    close(connessione_fd);
  }
}
```

Client

Client si collega al ServerC sulla porta 1025. Riceve la lista di negozi che sono disponibili ed effettua una scelta, inviandola al ServerC. Tramite menu sceglie varie operazioni tra cui esci, inserisci prodotto nel carrello, visualizza carrello e visualizza prodotti di un negozio.

```
#define PORTA 1025
int socket;                    // Socket connessione serverC
struct sockaddr_in indirizzo; // Indirizzo
printf("\e[2J\e[H\t\tBENVENUTO NEL MARKETPLACE");
// Creazione socket
socket = Socket(AF_INET, SOCK_STREAM, 0);
// Assegnazione indirizzo, porta, indirizzo
indirizzo.sin_family = AF_INET;
indirizzo.sin_port = htons(PORTA);
```

```
  indirizzo.sin_addr.s_addr = htonl(INADDR_ANY);
  // Connessione al serverC
  Connect(socket, (struct sockaddr *)&indirizzo, sizeof(indirizzo));
  struct Negozio *carrello = NULL; // Carello della spesa
  carrello = ricezioneListaNegozi(socket, carrello); // Ricevi lista dei negozi
  putchar('\n');
  int scelta; // Scelta client
  int flag;   // Flag controllo
  while (1) {
    printf("Scegli un operazione\n[0]Esci\n"
           "[1]Inserisci prodotto nel carello\n"
           "[2]Visualizza carello\n"
           "[3]Visualizza prodotti Negozio\n->");
    scanf("%d", &scelta);
    // Invia scelta al ServerC
    inviaCheck(socket, &scelta);
    printf("\n");
    switch (scelta) {
    // Esci
    case 0:
       ...
    // Inserisci prodotto nel carello
    case 1:
       ...
    // Visualizza carello
    case 2:
       ...
    // Visualizza prodotti di un negozio
    case 3:
       ...
    default:
       ...
    }
  }
  close(socket);
}
```

ClientN

ClientN si collega al ServerN sulla porta 1026. Inserisce le credenziali per accedere e le invia al ServerN. Dopo essere entrato sceglie il negozio su cui vuole operare. Successivamente viene stampato un menù con il quale il client interagisce per operare sul suo negozio. Le operazioni supportate sono: uscita dal programma, inserisci negozio, elimina negozio, gestione altro negozio, inserisci prodotto e elimina prodotto.

```
#define PORTA 1026
  int socket;                   // Socket connessione serverN
```

```c
struct sockaddr_in indirizzo; // Indirizzo
printf("\t\tBENVENUTO NEL MARKETPLACE\n");
// Creazione socket
socket = Socket(AF_INET, SOCK_STREAM, 0);
// Assegnazione indirizzo, porta, famiglia
indirizzo.sin_family = AF_INET;
indirizzo.sin_port = htons(PORTA);
indirizzo.sin_addr.s_addr = htonl(INADDR_ANY);
// Connessione ServerN
Connect(socket, (struct sockaddr *)&indirizzo, sizeof(indirizzo));
char username[25]; // Username utente
char pin[5];       // Pin utente
int check;         // Flag di controllo
int negozio;       // Id negozio
// Inserimento utente
   ...
// Riceve consenso dal Server N
riceviCheck(socket, &check);
if (!check) {
  printf("Credenziali sbagliate, disconnessione in corso...\n");
  close(socket);
  exit(0);
}
int scelta;
while (1) {
  printf("\nSu quale negozio vuoi operare: ");
  scanf("%d", &negozio);
  inviaProprietario(socket, username, &negozio);
  riceviCheck(socket, &check);
  if (check)
    break;
  printf("\nQuesto negozio non è di tua proprietà\n");
}
int flag = 1;         // Flag controlla eliminazione negozio corrente
struct Negozio *list; // Lista negozi gestiti da utente collegato
while (1) {
  printf("\e[2J\e[H \nI tuoi Negozi:\n");
  // Forza scelta per ricevere la lista di negozi
  scelta = 6;
  inviaCheck(socket, &scelta);
  inviaUsername(socket, username);
  list = ricezioneListaNegozi(socket, list);
  stampaLista(list);
  printf("\n");
  printf("Scegli un operazione\n"
         "[0]Esci\n"
```

```
            "[1]Inserisci prodotto\n"
            "[2]Elimina prodotto\n"
            "[3]Inserisci negozio\n"
            "[4]Elimina negozio\n"
            "[5]Gestisci altro negozio\n");
    // Se il negozio corrente non è stato eliminato
    if (flag) {...}
    // Se il negozio corrente è stato eliminato forza la scelta 5
    else {...}
    inviaCheck(socket, &scelta);
    char prodotto[25];   // Nome prodotto
    float prezzo;        // Prezzo prodotto
    int elimina_negozio; // Variabile controllo per l'eliminazione negozio
                         // corrente
    // Menu
    switch (scelta) {
    // Esci
    case 0:
       ...
    // Inserimento prodotto
    case 1:
      ...
    // Eliminazione prodotto
    case 2:
       ...
    // Inserimento negozio
    case 3:
       ...
    // Eliminazione negozio
    case 4:
       ...
    // Gestione altro negozio
    case 5:
       ...
    default:
       ...
    } // switch
  }   // while

  exit(0);
}
```

Manuale Utente

Per poter eseguire il programma e testarlo, bisogna scaricare ed estrarre la cartella **Marketplace.zip**.
Il progetto include 11 file: i file sorgente .c dei 3 Server e dei 2 client, più 3 file sorgente delle 3 librerie con i rispettivi 3 file header.
Per eseguire il programma è necessario compilare i 3 file dei Server e i 2 dei Client ed eseguirli da shell differenti.

Per compilare il ServerM

```
$ gcc -o serverM ServerM.c List.c Wrapped.c IO.c
```

Per Eseguire

```
$ ./serverM
```

Per compilare il ServerN

```
$ gcc -o serverN ServerN.c List.c Wrapped.c IO.c
```

Per Eseguire

```
$ ./serverN
```

Per compilare il ServerC

```
$ gcc -o serverC ServerC.c List.c Wrapped.c IO.c
```

Per Eseguire

```
$ ./serverC
```

Per compilare il Client

```
$ gcc -o client Client.c List.c Wrapped.c IO.c
```

Per Eseguire

```
$ ./client
```

Per compilare il ClientN

```
$ gcc -o clientN ClientN.c List.c Wrapped.c IO.c
```

Per Eseguire

```
$ ./clientN
```

Come primo passo si esegue prima il ServerM, poi i Server C ed N ed infine si aprono i client tutti in 5 terminali distinti.
Una volta eseguito il clientN possiamo testare il programma inserendo le credenziali:

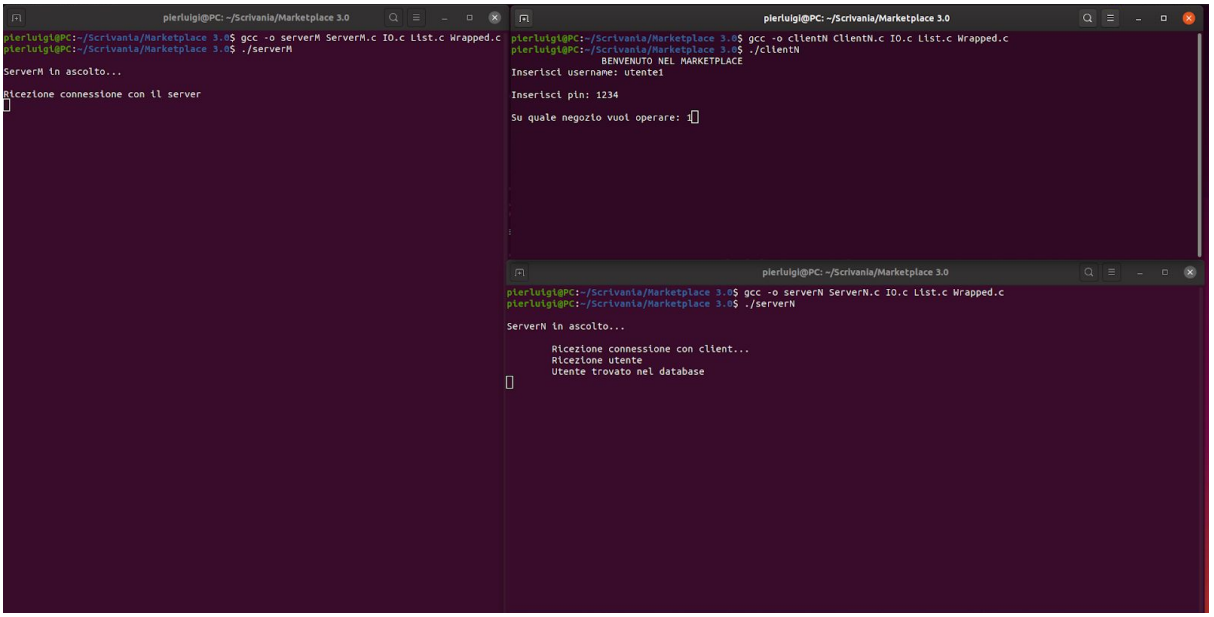
```
username: utente1
Pin: 1234
```

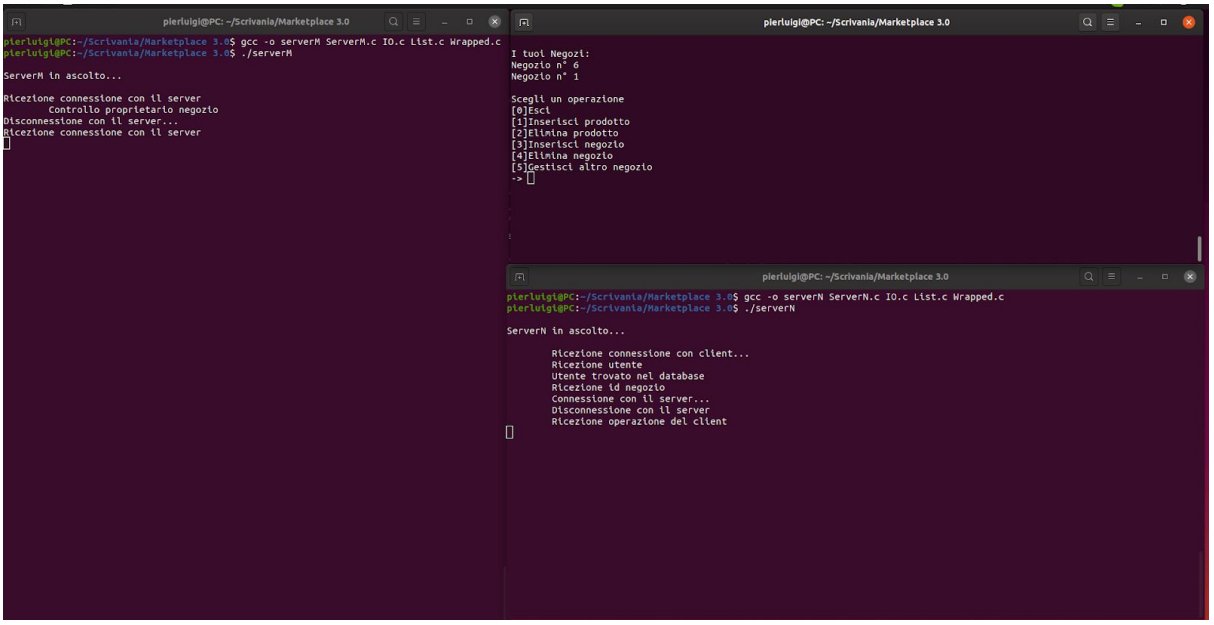
Sarà ora possibile interagire con i comandi dall'interfaccia del terminale. Si inseriscono di volta in volta il codice delle operazioni che si vogliono effettuare e infine si seguono le indicazioni stampate dal terminale.

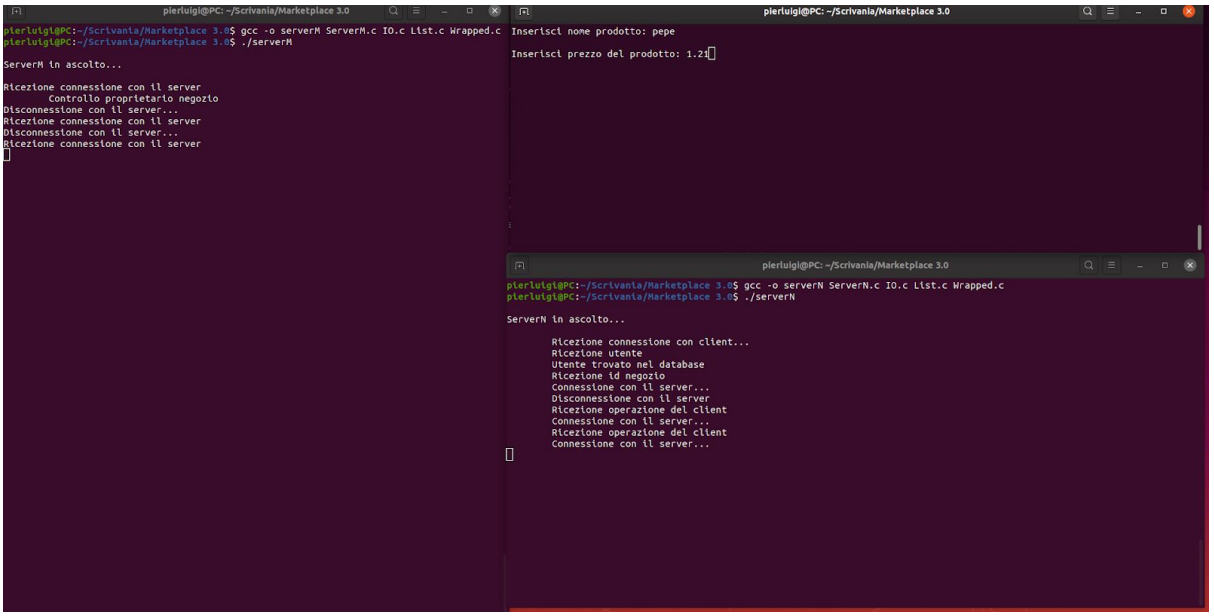
Esempio Esecuzione

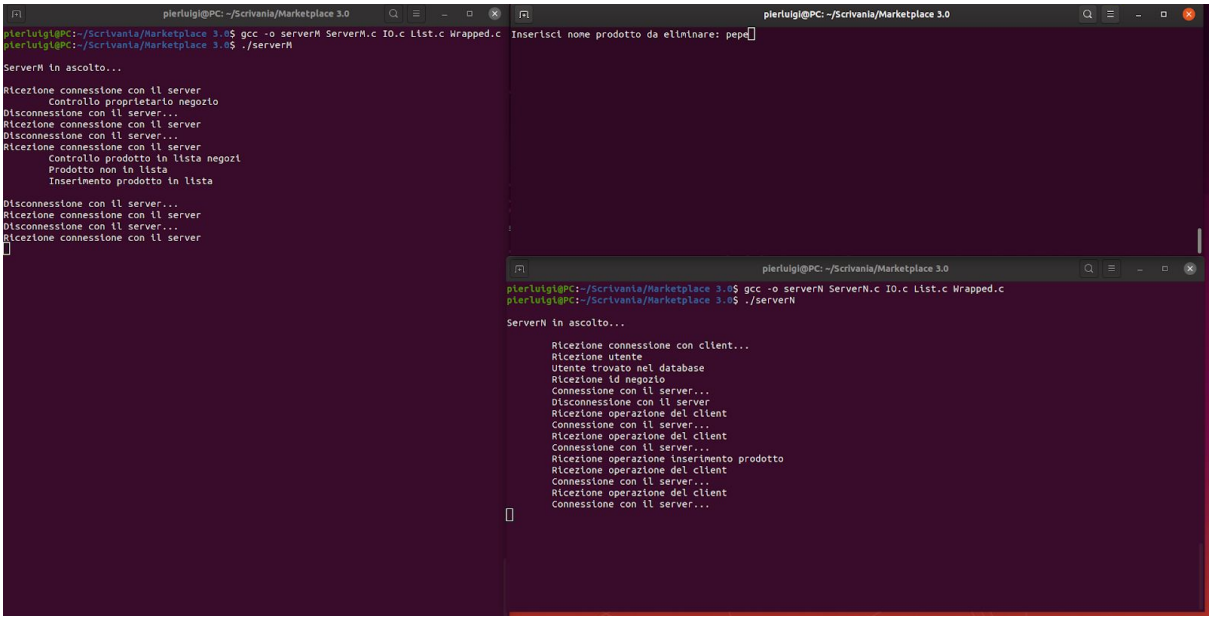
ClientN Esecuzione

Esempio di aggiunta e rimozione di un prodotto su clientN dopo aver effettuato l’accesso.

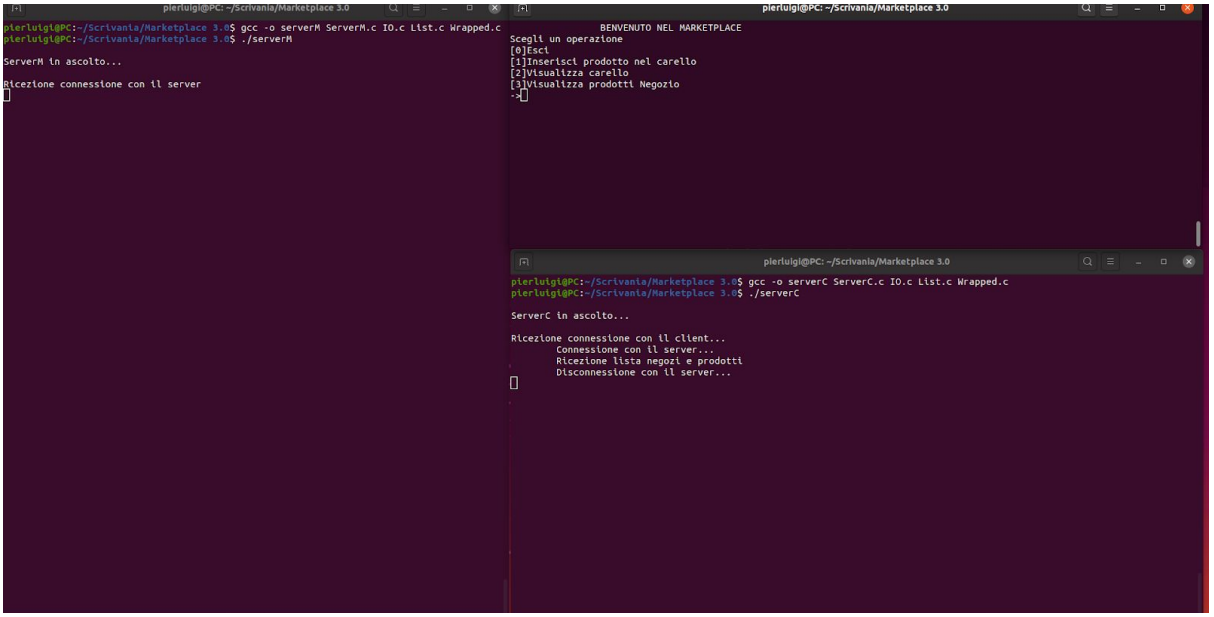


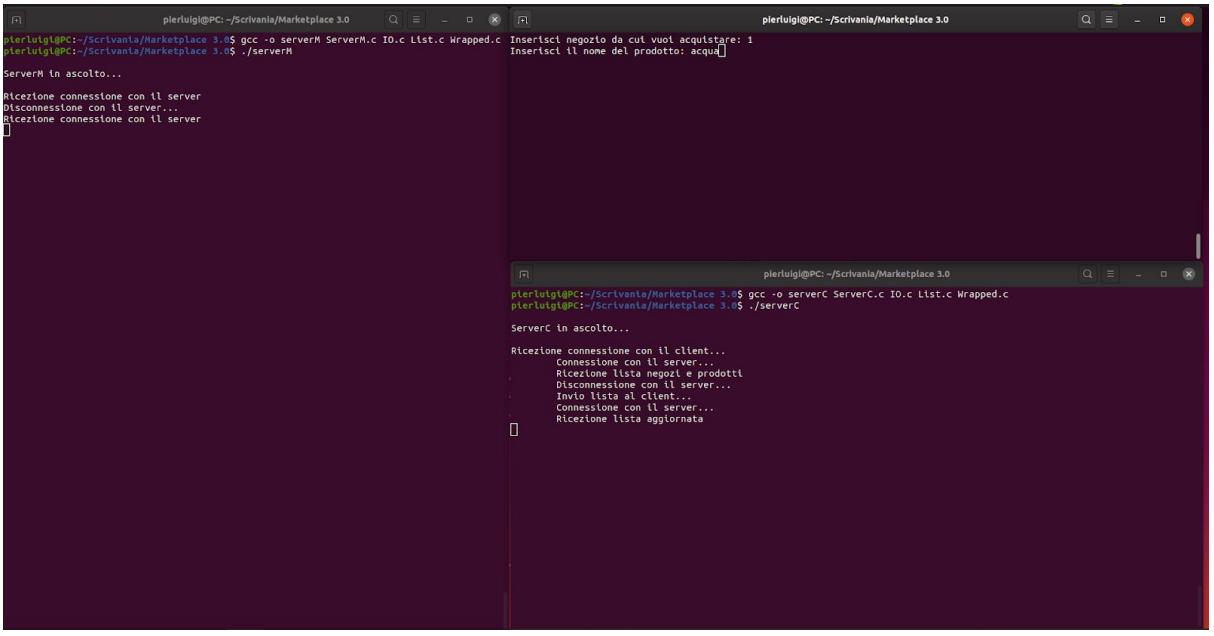


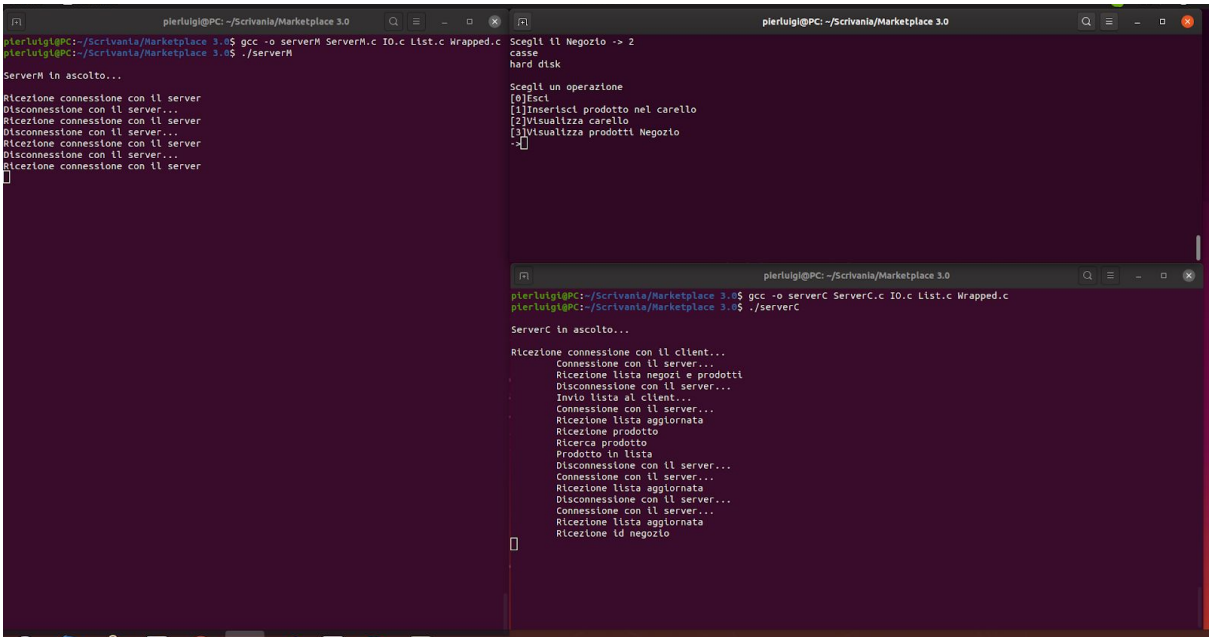

pierluigi@PC: ~/Scrivania/Marketplace 3.0
pierluigi@PC:~/Scrivania/Marketplace 3.0$ gcc -o serverM ServerM.c IO.c List.c Wrapped.c
pierluigi@PC:~/Scrivania/Marketplace 3.0$ ./serverM
ServerM in ascolto...
Ricezione connessione con il server
Controllo proprietario negozio
Disconnessione con il server...
Ricezione connessione con il server
Disconnesstione con il server...
Ricezione connesstione con il server
pierluigi@PC: ~/Scrivania/Marketplace 3.0
Inserisci nome prodotto: pepe
Inserisci prezzo del prodotto: 1.21
pierluigi@PC: ~/Scrivania/Marketplace 3.0
pierluigi@PC:~/Scrivania/Marketplace 3.0$ gcc -o serverN ServerN.c IO.c List.c Wrapped.c
pierluigi@PC:~/Scrivania/Marketplace 3.0$ ./serverN
ServerN in ascolto...
Ricezione connessione con client...
Ricezione utente
Utente trovato nel database
Ricezione id negozio
Connessione con il server...
Disconnessione con il server
Ricezione operazione del client
Connessione con il server...
Ricezione operazione del client
Connessione con il server...


pierluigi@PC: ~/Scrivania/Marketplace 3.0
pierluigi@PC:~/Scrivania/Marketplace 3.0$ gcc -o serverM ServerM.c IO.c List.c Wrapped.c
pierluigi@PC:~/Scrivania/Marketplace 3.0$ ./serverM
ServerM in ascolto...
Ricezione connessione con il server
Controllo proprietario negozio
Disconnessione con il server...
Ricezione connessione con il server
Disconnessione con il server...
Ricezione connessione con il server
Controllo prodotto in lista negozi
Prodotto non in lista
Inserimento prodotto in lista
Disconnessione con il server...
Ricezione connessione con il server
Disconnessione con il server...
Ricezione connessione con il server
pierluigi@PC: ~/Scrivania/Marketplace 3.0
Inserisci nome prodotto da eliminare: pepe
pierluigi@PC: ~/Scrivania/Marketplace 3.0
pierluigi@PC:~/Scrivania/Marketplace 3.0$ gcc -o serverN ServerN.c IO.c List.c Wrapped.c
pierluigi@PC:~/Scrivania/Marketplace 3.0$ ./serverN
ServerN in ascolto...
Ricezione connessione con client...
Ricezione utente
Utente trovato nel database
Ricezione id negozio
Connessione con il server...
Disconnessione con il server
Ricezione operazione del client
Connessione con il server...
Ricezione operazione del client
Connessione con il server...
Ricezione operazione inserimento prodotto
Ricezione operazione del client
Connessione con il server...
Ricezione operazione del client
Connessione con il server...

Client Esecuzione






pierluigi@PC: ~/Scrivania/Marketplace 3.0
pierluigi@PC:~/Scrivania/Marketplace 3.0$ gcc -o serverM ServerM.c IO.c List.c Wrapped.c
pierluigi@PC:~/Scrivania/Marketplace 3.0$ ./serverM
ServerM in ascolto...
Ricezione connessione con il server
Disconnessione con il server...
Ricezione connessione con il server
Disconnessione con il server...
Ricezione connessione con il server
Disconnessione con il server...
Ricezione connessione con il server
pierluigi@PC: ~/Scrivania/Marketplace 3.0
Scegli il Negozio -> 2
casse
hard disk
Scegli un operazione
[0]Esci
[1]Inserisci prodotto nel carello
[2]Visualizza carello
[3]Visualizza prodotti Negozio
->
pierluigi@PC: ~/Scrivania/Marketplace 3.0
pierluigi@PC:~/Scrivania/Marketplace 3.0$ gcc -o serverC ServerC.c IO.c List.c Wrapped.c
pierluigi@PC:~/Scrivania/Marketplace 3.0$ ./serverC
ServerC in ascolto...
Ricezione connessione con il client...
Connessione con il server...
Ricezione lista negozi e prodotti
Disconnessione con il server...
Invio lista al client...
Connessione con il server...
Ricezione lista aggiornata
Ricezione prodotto
Ricerca prodotto
Prodotto in lista
Disconnessione con il server...
Connessione con il server...
Ricezione lista aggiornata
Disconnessione con il server...
Connessione con il server...
Ricezione lista aggiornata
Ricezione id negozio