UNIVERSITÀ DEGLI STUDI DI NAPOLI "PARTHENOPE"
FACOLTÀ DI SCIENZE E TECNOLOGIE

CORSO DI LAUREA IN INFORMATICA (PERCORSO GENERALE)

RELAZIONE DI RETI DEI CALCOLATORI

DOCENTE
Prof. Alessio Ferone

CANDIDATO
Luca Esposito
matricola 0124001698

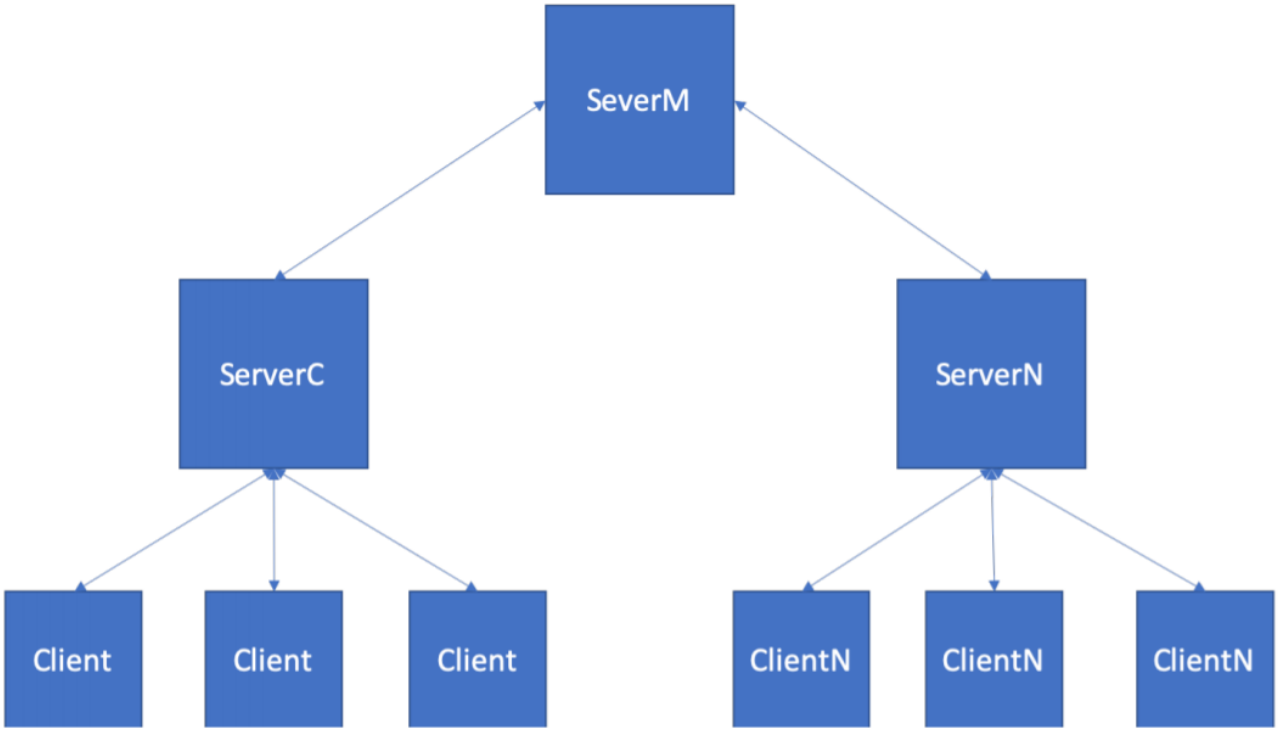
ANNO ACCADEMICO 2019-2020

INDICE

1. Descrizione del progetto

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 può 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.



2. Descrizione e schemi dell'architettura

Il modello di programmazione scelto per la realizzazione del progetto è quello client-server, che è caratterizzato da due soggetti: un server cioè un programma che riceve richieste e fornisce servizi e un client che è un programma che invia le richieste al server e riceve i servizi da esso.

Il server utilizzato è concorrente ovvero è in grado di fornire servizi a più client contemporaneamente, ciò è effettuato attraverso l'I/O Multiplex da parte del server, il quale ha il compito di fornire i servizi richiesti.

Nel modello di I/O multiplex il processo resta in attesa di eventi su uno o più descrittori e l'esecuzione si blocca fino a quando uno dei descrittori diventa pronto. Il vantaggio nell'uso di questo modello è che si possono monitorare più canali di comunicazione.

L'implementazione è possibile attraverso la funzione select() che comunica al kernel di monitorare un insieme di descrittori, ponendo il processo in waiting e risvegliandolo quando si verifica un evento.

In sintesi:

a) ServerM ha una socket che riceve richieste da ServerN e ServerC e gestirà una lista (nella lista negozi virtuali è presente una lista prodotti) su cui effettuare ricerca, inserimento ed eliminazione.

b) ServerN avrà una socket su ServerM e una su ClientN e invierà richieste al ServerM di eliminazione e inserimento (sulle liste negozi virtuali e prodotti).
c) ServerC avrà una socket su ServerM e una su Client e invierà richieste al ServerM di ricerca e elenco (di negozi virtuali e prodotti).

d) ClientN avrà una socket su ServerN e consente di gestire i negozi virtuali.
e) Client avrà una socket su ServerC e consente di inserire prodotti di diversi negozi virtuali in una lista degli acquisti e di visualizzare la lista degli acquisti.

Richieste al ServerM:

- Da ServerN:
  - eliminazione_prodotto
  - inserimento_prodotto
  - eliminazione_negozio
  - inserimento_negozio
- Da ServerC:
  - ricerca_prodotto
  - elenco_prodotti_negozio
  - ricerca_negozio
  - elenco_negozi
  - inserimento_in_lista_acquisti

Per rendere l'esecuzione automatica, sono state effettuate le seguenti #define:

```
#define IP_SERVER_M "127.0.0.1"
#define PO_SERVER_M 1025

#define IP_SERVER_N "127.0.0.1"
#define PO_SERVER_N 1026

#define IP_SERVER_C "127.0.0.1"
#define PO_SERVER_C 1027
```

Struttura dei pacchetti:

Ogni pacchetto è associato alla struct Pacchetto la quale sarà così composta:

```
typedef struct pacchetto_prodotto
{
    int n_richiesta;

    //Prodotto
    char nome_prodotto[len_stringa];
    float prezzo;

    //Negozio
    char nome_negozio[len_stringa];
}Pacchetto;
```

Struttura dati (Lista):

- La struttura dati Lista verrà implementata come lista bidirezionale e sarà formata dagli n nodi che comporranno i negozi.
- Ogni negozio avrà al suo interno una lista bidirezionale che conterrà i prodotti del negozio.
- La lista dei negozi è mantenuta solo all'interno del ServerM

```
typedef struct nodo_prodotto
{
    char nome_prodotto[len_stringa];
    float prezzo;
    struct nodo_prodotto *prev, *next;
}Nodo_prodotto;

typedef struct nodo_negozio
{
    char nome_negozio[len_stringa];
    Nodo_prodotto *Lista_prodotti_head, *Lista_prodotti_coda, *Lista_prodotti_work;

    struct nodo_negozio *prev, *next;
}Nodo_negozio;
```

- `Nodo_negozio* ricerca_negozio(char nome_negozio[len_stringa]);`
  Ricerca un negozio (attraverso nome_negozio) e tornerà NULL nel caso non sia presente.

- `Nodo_prodotto* ricerca_prodotto(char nome_prodotto[len_stringa]);`
  Ricerca un prodotto all'interno dell'intera lista negozi. Ritornerà NULL nel caso non sia presente.

- `int inserimento_negozio(char nome_negozio[len_stringa]);`
  Inserisce un nuovo negozio nella lista negozi. Ritorna -1 nel caso in cui il negozio fosse già presente, altrimenti ritornerà 0.

- `int inserimento_prodotto(char nome_negozio[len_stringa], char nome_prodotto[len_stringa], float prezzo);`
  Inserisce un prodotto nel negozio passato come argomento. Ritorna -1 se il prodotto è già presente oppure il negozio non esiste, altrimenti ritornerà 0.

- `Nodo_prodotto* inserimento_in_lista(char nome_prodotto[len_stringa], Nodo_prodotto *L);`
  Per il ClientC, inserisce un nuovo prodotto in coda alla lista L degli acquisti. Ritornerà l'ultimo elemento della lista inserito.

- `int eliminazione_negozio(char nome_negozio[len_stringa]);`
  Elimina un negozio dalla lista negozi nel ServerM. Ritorna -1 se il negozio non esiste o la lista risulta essere vuota, altrimenti ritornerà 0.

- `int eliminazione_prodotto(char nome_prodotto[len_stringa]);`
  Elimina un prodotto (ricercandolo in tutti i negozi). Ritorna -1 se il prodotto non esiste, altrimenti ritorna 0.

3. Descrizione e schemi del protocollo applicazione

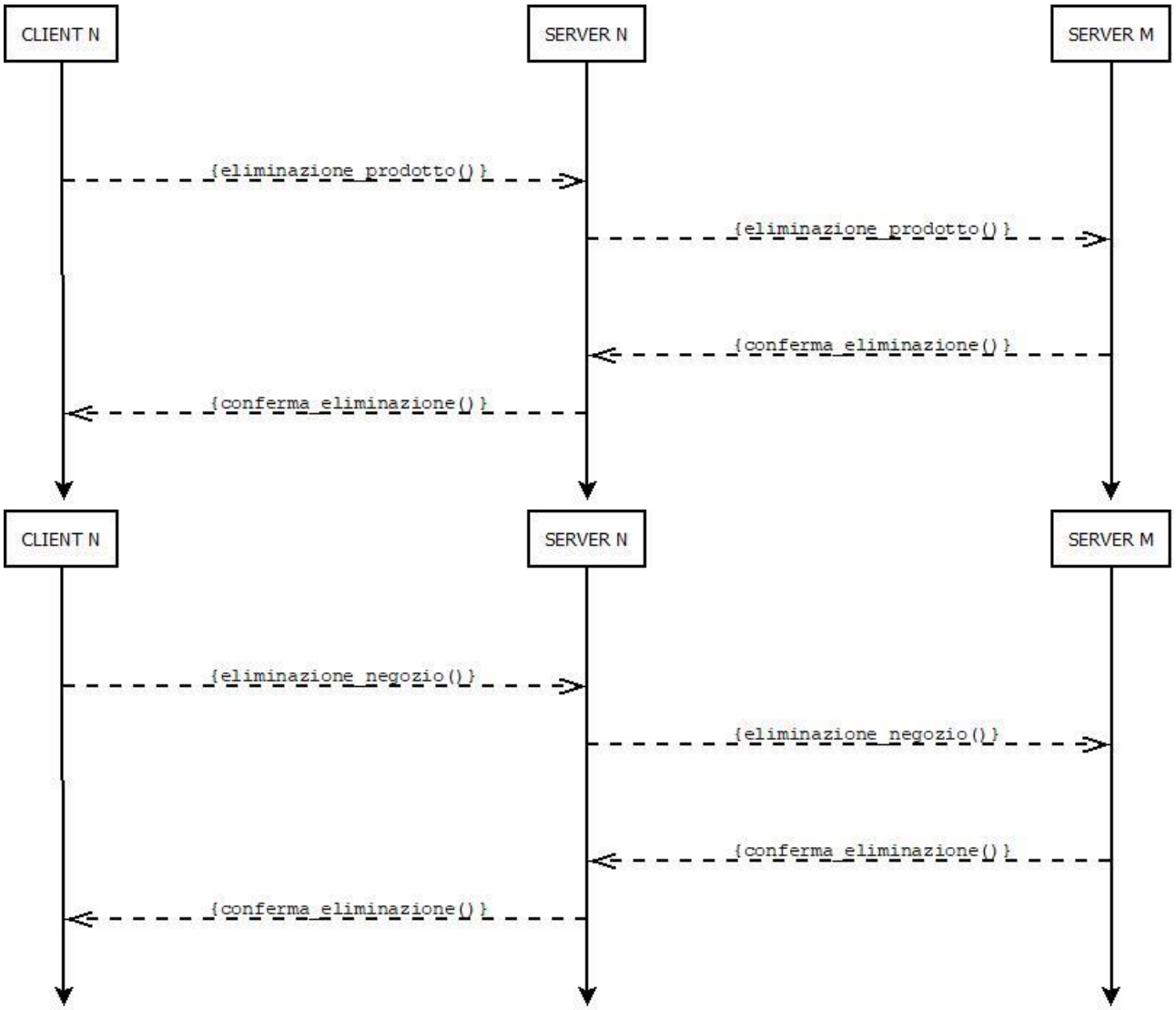
Questo progetto è scritto in linguaggio C sotto piattaforma UNIX.

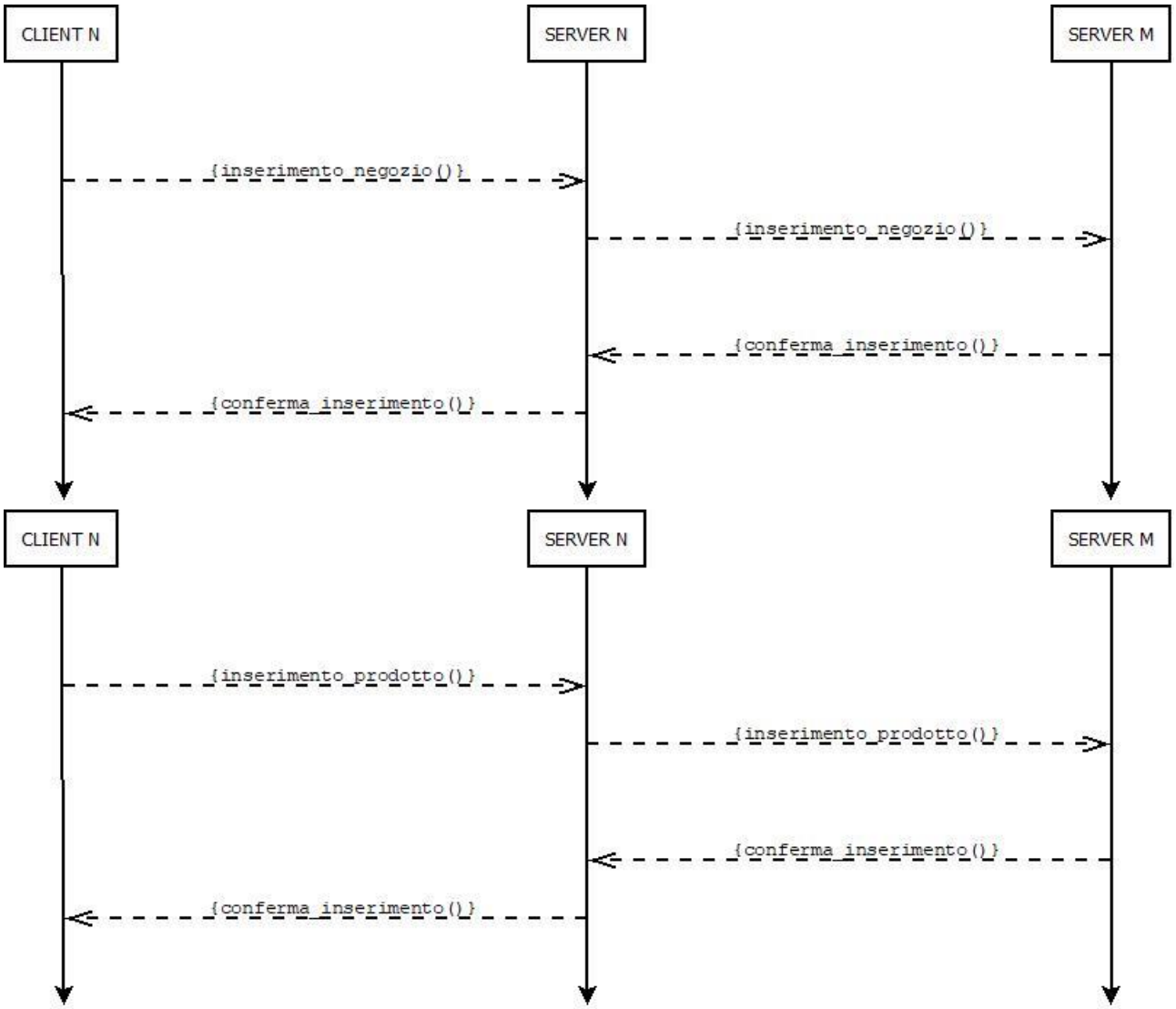
Il tipo di socket utilizzato in questa applicazione è una streaming socket (SOCK_STREAM), ciò vuol dire che essa fornisce una connessione sequenziale, affidabile e full-duplex. Inoltre si può osservare che è stata utilizzata "l'accoppiata" AF_INET+SOCK_STREAM, che determina una connessione TCP.

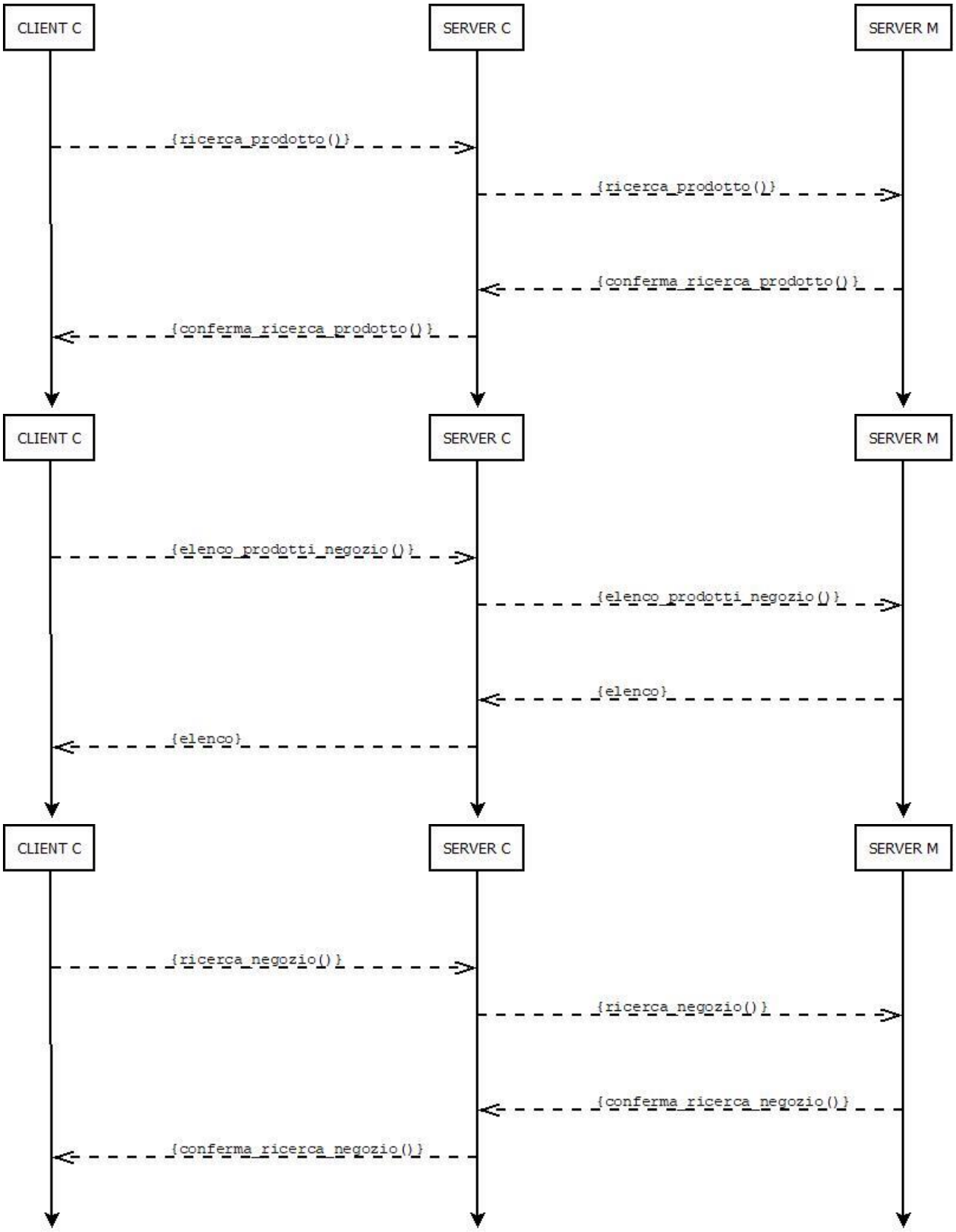
Il protocollo TCP socket, essendo orientato alla connessione, prima di poter trasmettere dati, deve stabilire la comunicazione, negoziando una connessione tra mittente e destinatario (che viene esplicitamente chiusa quando non più necessaria). In tal modo si ha la garanzia che il server riceva le "scelte" inviate dal client e si comporti di conseguenza.

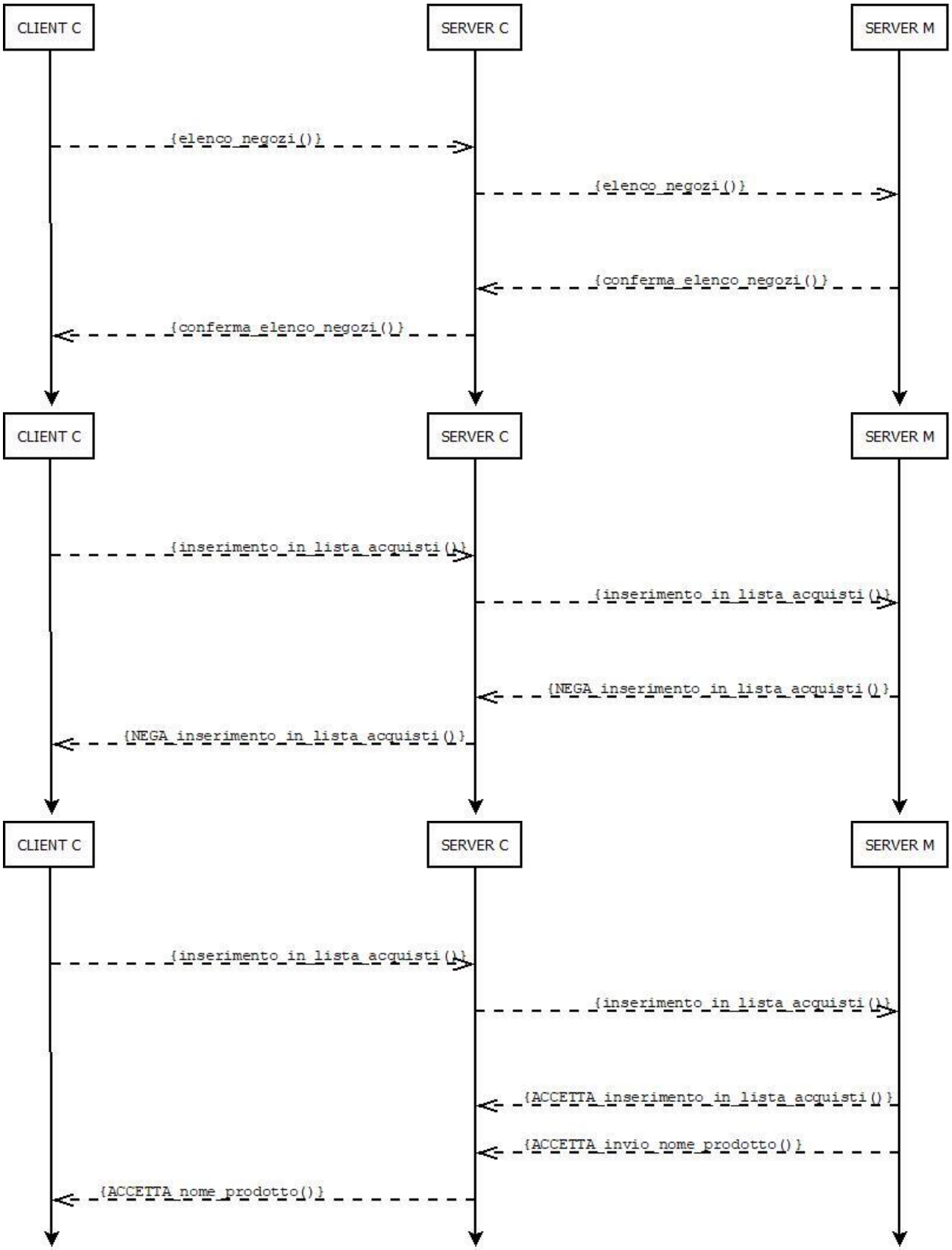
Schema delle fasi:

1. Client (N o C) effettua richiesta al Server (rispettivamente N o C)
2. Richiesta ricevuta dal Server (N o C) che apre la comunicazione con il ServerM
3. Server (N o C) inoltra richiesta al ServerM e resta in attesa di risposta
4. ServerM risponde al Server (N o C)
5. Server (N o C) invia risposta la Client (rispettivamente N o C)




CLIENT N
SERVER N
SERVER M
{inserimento_negozio()}
{inserimento_negozio()}
{conferma_inserimento()}
{conferma_inserimento()}
CLIENT N
SERVER N
SERVER M
{inserimento_prodotto()}
{inserimento_prodotto()}
{conferma_inserimento()}
{conferma_inserimento()}


CLIENT C
SERVER C
SERVER M
{ricerca_prodotto()}
{ricerca_prodotto()}
{conferma_ricerca_prodotto()}
{conferma_ricerca_prodotto()}
CLIENT C
SERVER C
SERVER M
{elenco_prodotti_negozio()}
{elenco_prodotti_negozio()}
{elenco}
{elenco}
CLIENT C
SERVER C
SERVER M
{ricerca_negozio()}
{ricerca_negozio()}
{conferma_ricerca_negozio()}
{conferma_ricerca_negozio()}


CLIENT C
SERVER C
SERVER M
{elenco_negozi()}
{elenco_negozi()}
{conferma_elenco_negozi()}
{conferma_elenco_negozi()}
CLIENT C
SERVER C
SERVER M
{inserimento_in_lista_acquisti()}
{inserimento_in_lista_acquisti()}
{NEGA_inserimento_in_lista_acquisti()}
{NEGA_inserimento_in_lista_acquisti()}
CLIENT C
SERVER C
SERVER M
{inserimento_in_lista_acquisti()}
{inserimento_in_lista_acquisti()}
{ACCETTA_inserimento_in_lista_acquisti()}
{ACCETTA_invio_nome_prodotto()}
{ACCETTA_nome_prodotto()}

4. Dettagli implementativi del ClientN

Il ClientN permette operazioni di:

1. eliminazione_prodotto
2. inserimento_prodotto
3. eliminazione_negozio
4. inserimento_negozio

Viene creata una socket TCP ed IPv4 con AF_INET + SOCK_STREAM e la Connect() viene effettuata sul ServerN.

Il ciclo while() permette di non chiudere il client (e la connessione al server) finchè l'utente non darà in input 0.

La select() permette di controllare due file descriptor (STDIN_FILENO e sockfd). Quando uno di questi due file descriptor risulterà attivo, la select() sbloccherà l'esecuzione e si entrerà nei due IF:

- if(FD_ISSET(STDIN_FILENO, &set)) permette di controllare l'input sullo stdin e invia il pacchetto al ServerN
- if(FD_ISSET(sockfd, &set)) permette di controllare il file descriptor del sockfd da cui si riceverà la risposta dal ServerN.

5. Dettagli implementativi del ClientC

Il ClientC permette operazioni di:

5. ricerca_prodotto
6. elenco_prodotti_negozio
7. ricerca_negozio
8. elenco_negozi
9. inserimento_in_lista
10. visualizza_lista_acquisti

Viene creata una socket TCP ed IPv4 con AF_INET + SOCK_STREAM e la Connect() viene effettuata sul ServerC.

Il ClientC gestisce una lista bidirezionale propria per i prodotti nella lista acquisti.

Il ciclo while() permette di non chiudere il client (e la connessione al server) finchè l'utente non darà in input 0.

La select() permette di controllare due file descriptor (STDIN_FILENO e sockfd). Quando uno di questi due file descriptor risulterà attivo, la select() sbloccherà l'esecuzione e si entrerà nei due IF:

- if(FD_ISSET(STDIN_FILENO, &set)) permette di controllare l'input sullo stdin e invia il pacchetto al ServerC
- if(FD_ISSET(sockfd, &set)) permette di controllare il file descriptor del sockfd da cui si riceverà la risposta dal ServerC.
    - Se la prima FullRead dovesse essere "TRUE" si entra nel secondo IF che gestisce una seconda FullRead che conterrà il nome del prodotto da inserire nella lista degli acquisti dell'utente

6. Dettagli implementativi del ServerN

Il ServerN riceve pacchetti di:

1. eliminazione_prodotto
2. inserimento_prodotto
3. eliminazione_negozio
4. inserimento_negozio

- ✓ Viene creata una socket TCP ed IPv4 con AF_INET + SOCK_STREAM.
- ✓ `Setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &(int){ 1 }, sizeof(int));` permette di riutilizzare l'indirizzo
- ✓ `struct sockaddr_in servaddr` conterrà l'indirizzo IP "127.0.0.1" e la porta "1026"

L'implementazione del Server è effettuata mediante I/O Multiplex, in particolare il ServerN manterrà una connessione sugli n client tramite i file descriptor e, una volta ricevuto un pacchetto da un ClientN, aprirà una connessione sul ServerM, fungendo da client, ricevendo risposta che poi invierà al ClientN.

La select() permette di controllare il file descriptor della socket del server (sockfd). Quando si verifica un evento, la select() sbloccherà l'esecuzione e si entrerà nell'IF in cui verrà effettuata l'Accept() sul ClientN.

Il `while(n != 0)` ci permette di servire tutte le connessioni dei client associate all'i-esimo file descriptor.

Successivamente, in `if(FD_ISSET(i, &set))` viene servito il client i-esimo dove, il ServerN, crea una connessione con il ServerM fungendo da client. In questo modo, il ServerN invierà la richiesta ricevuta dal ClientN al ServerM, attenderà risposta e risponderà al ClientN.

7. Dettagli implementativi del ServerC

Il ServerC riceve pacchetti di:

5. ricerca_prodotto
6. elenco_prodotti_negozio
7. ricerca_negozio
8. elenco_negozi
9. inserimento_in_lista

- ✓ Viene creata una socket TCP ed IPv4 con AF_INET + SOCK_STREAM.
- ✓ `Setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &(int){ 1 }, sizeof(int));` permette di riutilizzare l'indirizzo
- ✓ `struct sockaddr_in servaddr` conterrà l'indirizzo IP "127.0.0.1" e la porta "1027"

L'implementazione del Server è effettuata mediante I/O Multiplex, in particolare il ServerC manterrà una connessione sugli n client tramite i file descriptor e, una volta ricevuto un pacchetto da un ClientC, aprirà una connessione sul ServerM, fungendo da client, ricevendo risposta che poi invierà al ClientC.

La select() permette di controllare il file descriptor della socket del server (sockfd). Quando si verifica un evento, la select() sbloccherà l'esecuzione e si entrerà nell'IF in cui verrà effettuata l'Accept() sul ClientC.

Il `while(n != 0)` ci permette di servire tutte le connessioni dei client associate all'i-esimo file descriptor.

Successivamente, in `if(FD_ISSET(i, &set))` viene servito il client i-esimo dove, il ServerC, crea una connessione con il ServerM fungendo da client. In questo modo, il ServerC invierà la richiesta ricevuta dal ClientC al ServerM, attenderà risposta e risponderà al ClientC.

- ➢ A differenza del ServerN, il ServerC effettua un controllo sulla prima FullRead effettuata sulla socket del ServerM. Se `if(strcmp(risposta, "TRUE") == 0)` il ClientC si aspetterà una seconda FullWrite (ossia farà una FullRead) per ricevere il nome del prodotto da inserire nella lista degli acquisti.

8. Dettagli implementativi del ServerM

Il ServerM riceve pacchetti di:

1. eliminazione_prodotto
2. inserimento_prodotto
3. eliminazione_negozio
4. inserimento_negozio
5. ricerca_prodotto
6. elenco_prodotti_negozio
7. ricerca_negozio
8. elenco_negozi
9. inserimento_in_lista

- ✓ Viene creata una socket TCP ed IPv4 con AF_INET + SOCK_STREAM.
- ✓ `Setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &(int){ 1 }, sizeof(int));` permette di riutilizzare l'indirizzo
- ✓ `struct sockaddr_in servaddr` conterrà l'indirizzo IP "127.0.0.1" e la porta "1027"

L'implementazione del Server è effettuata mediante I/O Multiplex, in particolare il ServerM manterrà una connessione sui due Server (N e C) ad ogni richiesta. Dunque la connessione verrà aperta e chiusa ogni volta che ci sarà una richiesta, poiché, il ServerM, non deve attendere, bensì elaborare la richiesta e dare subito una risposta. L'attesa al client è delegata ai ServerN e ServerC.

La select() permette di controllare il file descriptor della socket del server (sockfd). Quando si verifica un evento, la select() sbloccherà l'esecuzione e si entrerà nell'IF in cui verrà effettuata l'Accept() sul ServerC o ServerN.

Il `while(n != 0)` ci permette di servire tutte le connessioni dei client associate all'i-esimo file descriptor.

Successivamente, in `if(FD_ISSET(i, &set))` viene servito il client i-esimo (ServerN o ServerC che si comportano da client) dove, il ServerM, riceve il pacchetto, lo elabora (attraverso lo switch) ed invia la risposta al ServerC o ServerN.

9. Manuale utente

In questo capitolo, si descrivono brevemente i files contenuti nel file Progetto_Reti_Esposito_Luca.zip. Il pacchetto contiene i seguenti files:

- ✓ Relazione_Reti_Esposito_Luca.pdf
- ✓ header.h
- ✓ header_fun.h
- ✓ ServerM.c
- ✓ ServerN.c
- ✓ ServerC.c
- ✓ clientN.c
- ✓ ClientC.c
- ✓ Script.sh

Per verificare l'effettivo funzionamento dell'applicazione bisogna compilare e eseguire in UNIX attraverso terminale, in ordine:

1. gcc ServerM.c -o ServerM.out
2. ./ServerM.out

3. gcc ServerN.c -o ServerN.out
4. ./ServerN.c

5. gcc ServerC.c -o ServerC.out
6. ./ServerC.out

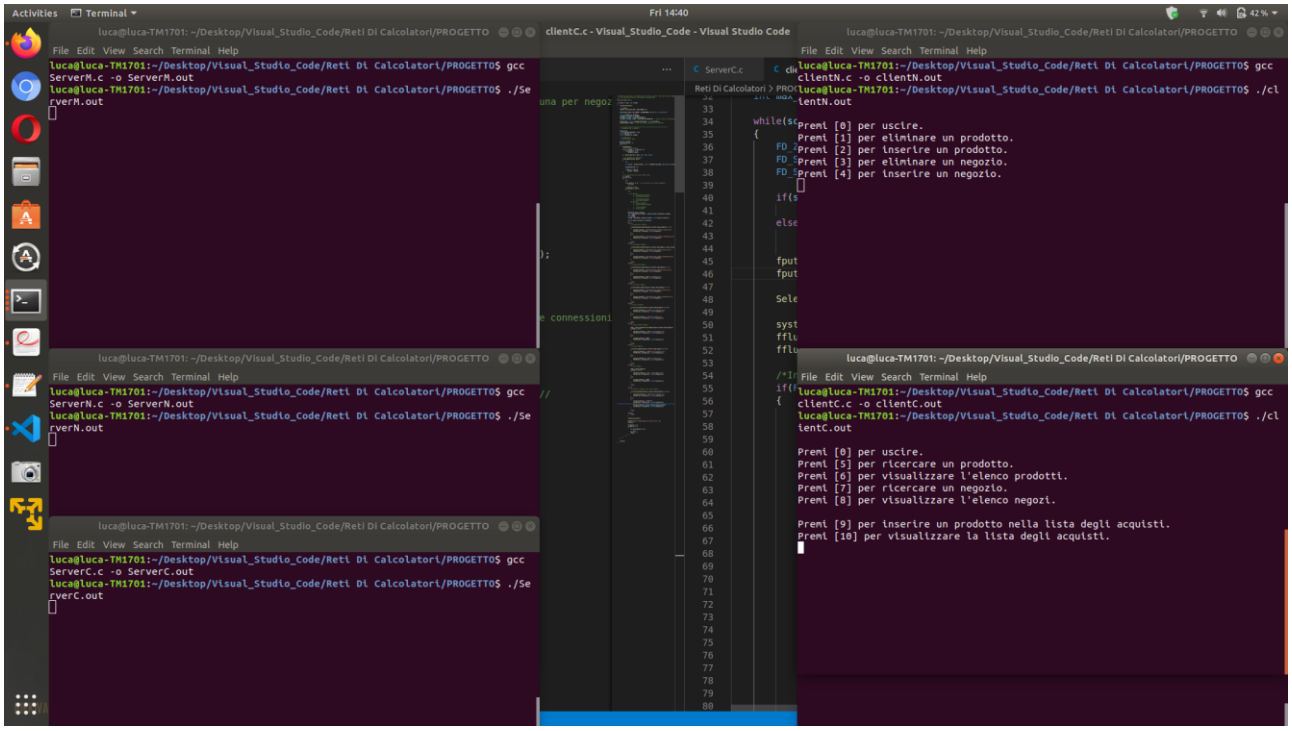
7. gcc clientN.c -o clientN.out
8. ./clientN.out

9. gcc clientC.c -o clientC.out
10. ./clientC.out

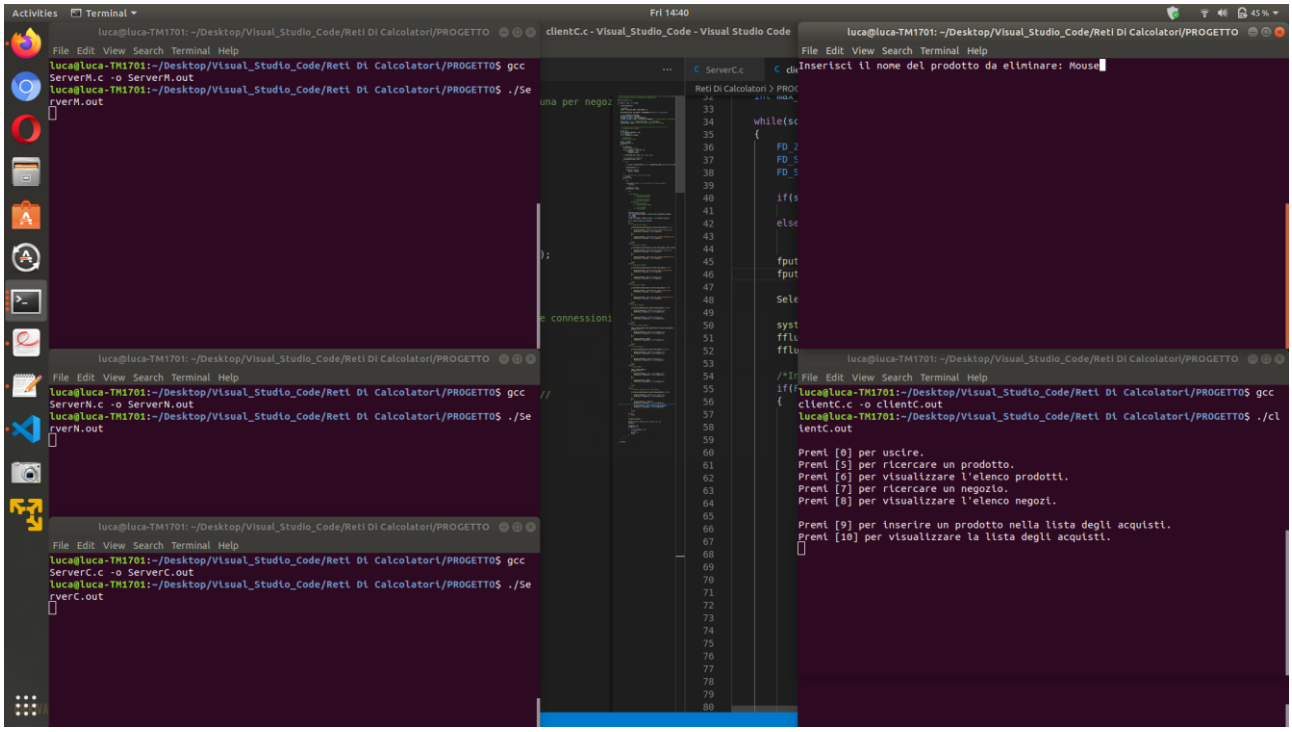
Una volta eseguite le varie fasi, l'utente si interfaccerà con il clientC e il clientN.

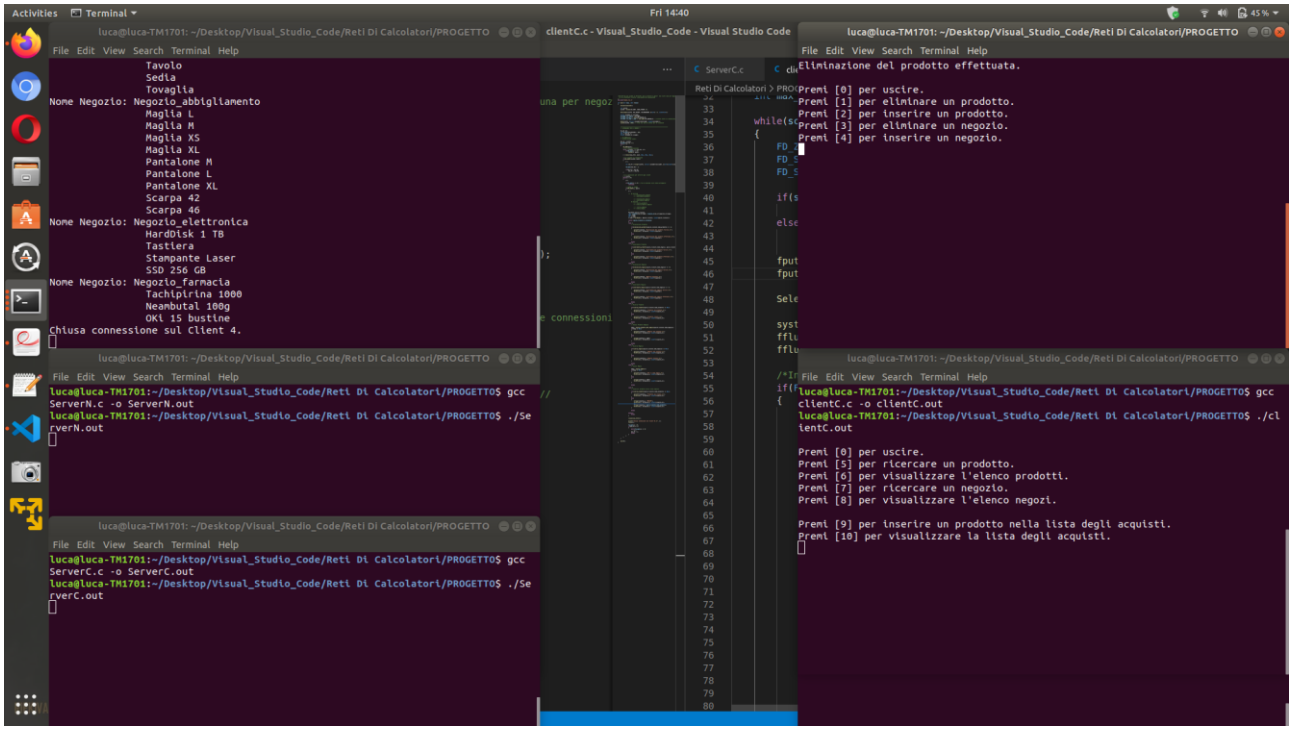
Sono state rese tutte le operazioni di compilazione ed esecuzione automatiche tramite il file script.sh. Basterà avviare da terminale il file con ./script.sh

A. Esecuzione iniziale: si compilano e avviano in successione ServerM, ServerN, ServerC. I ClientN e ClientC possono essere compilati e avviati solo successivamente. I Server possono gestire più client mediante I/O Multiplex.

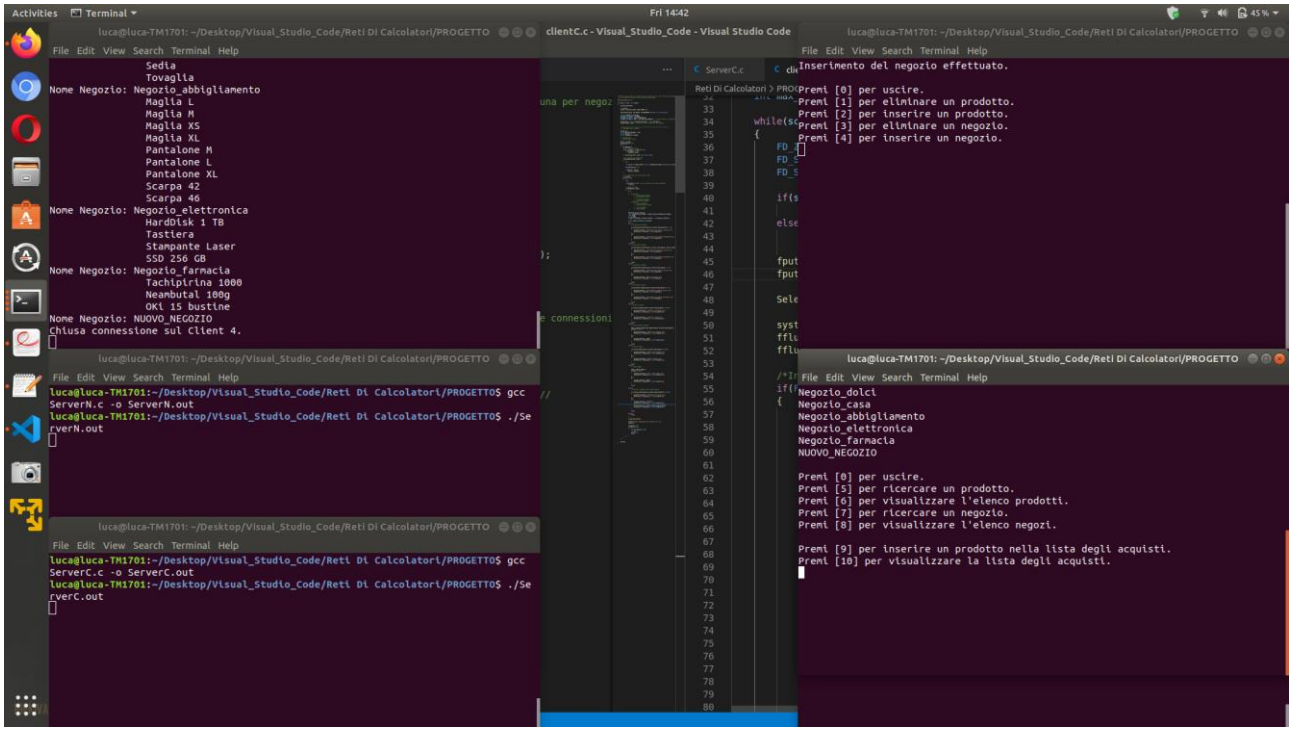


B. Esempio di eliminazione di un prodotto: viene selezionata l'opzione e inserito il nome del prodotto da eliminare. Una volta dato "INVIO" il prodotto verrà eliminato ed il client riceverà un messaggio di avvenuta eliminazione mentre il ServerM visualizzerà la lista di tutti i negozi e prodotti disponibili.

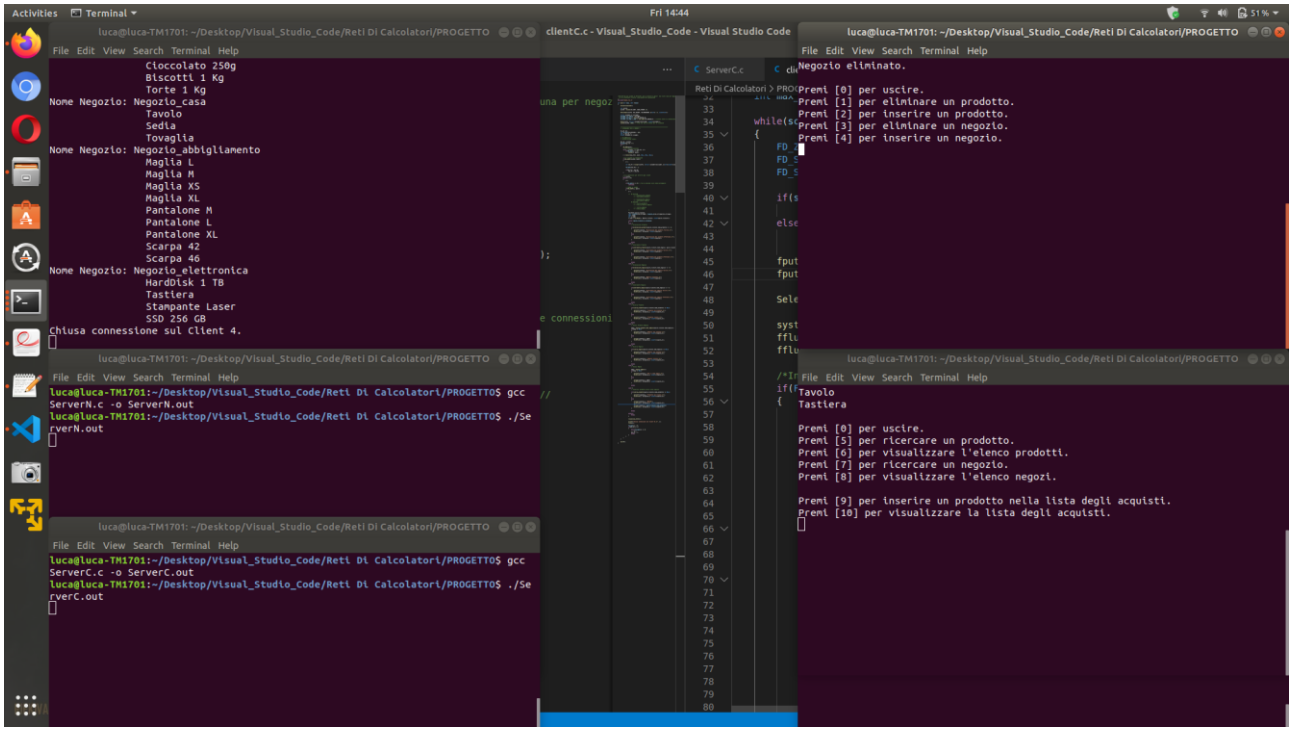




C. Esempio di inserimento di un negozio da ClientN e visualizzazione dei negozi disponibili da ClientC: sul ClientN viene selezionata l'opzione [4], l'utente inserirà il nome del negozio e, una volta premuto "INVIO", il ServerM visualizzerà tutto. Successivamente, sul ClientC, viene selezionata l'opzione [8] che permetterà di visualizzare tutti i negozi disponibili, incluso il nuovo negozio inserito dal ClientN.



D. Esempio di eliminazione del Negozio_farmacia e visualizzazione della lista degli acquisti del ClientC: dal ClientN viene selezionata l'opzione [3] per eliminare il negozio, successivamente il ServerM visualizza la lista aggiornata. Dal ClientC vengono effettuate due opzioni [9] dove vengono inseriti nella lista degli acquisti "Tastiera" e "Tavolo".



10. Codice Sorgente

FILE: header.h

Il file “header.h” contiene gli #include alle librerie, le funzioni FullRead() e FullWrite() e le funzioni wrapper.

```
1.  #include<stdio.h>
2.  #include<stdlib.h>
3.  #include<string.h>
4.  #include<unistd.h>
5.  #include<netdb.h>
6.  #include<errno.h>
7.  #include<time.h>
8.  #include<limits.h>
9.  #include<arpa/inet.h>
10. #include<sys/socket.h>
11. #include<sys/select.h>
12.
13. int FullWrite(int fd, void *buf, int count)
14. {
15.     int n_written, n_left = count;
16.
17.     while(n_left > 0)
18.     {
19.         if((n_written = write(fd, buf, n_left)) < 0)
20.         {
21.             if(errno == EINTR)
22.                 continue;
23.             else
24.                 exit(n_written);
25.         }
26.
27.         n_left -= n_written;
28.         buf += n_written;
29.     }
30.
31.     return n_left;
32. }
33. int FullRead(int fd, void *buf, int count)
34. {
35.     int n_read, n_left = count;
36.
37.     while(n_left > 0)
38.     {
39.         if((n_read = read(fd, buf, n_left)) < 0)
40.         {
41.             if(errno == EINTR)
42.                 continue;
43.             else
44.                 exit(n_read);
45.         }
46.         else if(n_read == 0)
47.             break;
48.
49.         n_left -= n_read;
50.         buf += n_read;
51.     }
52.
53.     buf = 0;
54.     return n_left;
55. }
56.
57. int Socket(int domain, int type, int protocol)
58. {
```

```
59.     int sockfd;
60.     if((sockfd = socket(domain, type, protocol)) < 0)
61.     {
62.         perror("!SOCKET!");
63.         exit(-1);
64.     }
65.
66.     return sockfd;
67. }
68.
69. //CLIENT
70. struct hostent* Gethostbyname(char *argv)
71. {
72.     struct hostent *simbolicAddress;
73.     if((simbolicAddress = gethostbyname(argv)) == NULL)
74.     {
75.         perror("!GET_HOST_BY_NAME!");
76.         exit(-1);
77.     }
78.
79.     return simbolicAddress;
80. }
81.
82. void Connect(int sockfd, const struct sockaddr* addr, socklen_t addrlen)
83. {
84.     if(connect(sockfd, addr, addrlen) < 0)
85.     {
86.         perror("!CONNECT!");
87.         exit(-1);
88.     }
89. }
90.
91. //SERVER
92. void Setsockopt(int sockfd, int level, int optname, void* val, socklen_t size_of)
93. {
94.     if(setsockopt(sockfd, level, optname, val, size_of) < 0)
95.     {
96.         perror("!SET_SOCK_OPT!");
97.         exit(-1);
98.     }
99. }
100.
101.        void Bind(int sockfd, const struct sockaddr* addr, socklen_t addrlen)
102.        {
103.            if(bind(sockfd, addr, addrlen) < 0)
104.            {
105.                perror("!BIND!");
106.                exit(-1);
107.            }
108.        }
109.
110.        void Listen(int sockfd, int n)
111.        {
112.            if(listen(sockfd, n) < 0)
113.            {
114.                perror("!LISTEN!");
115.                exit(-1);
116.            }
117.        }
118.
119.        int Accept(int sockfd, struct sockaddr* addr, socklen_t* addrlen)
120.        {
121.            int new_fd;
122.            if((new_fd = accept(sockfd, addr, addrlen)) < 0)
123.            {
124.                perror("!ACCEPT!");
125.                exit(-1);
```

```
126.            }
127.
128.            return new_fd;
129.        }
130.
131.        int Select(int max_fd, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct
    timeval *timeout)
132.        {
133.            int n;
134.            if((n = select(max_fd, readfds, writefds, exceptfds, timeout)) < 0)
135.            {
136.                perror("!SELECT!");
137.                exit(-1);
138.            }
139.
140.            return n;
141.        }
```

FILE: header_fun.h

Il file “header_fun.c” contiene la struttura dati Nodo_negozio, Nodo_prodotto, Pacchetto e le function collegate.

```
1.  #include"header.h"
2.  #define len_stringa 50
3.
4.  #define IP_SERVER_M "127.0.0.1"
5.  #define PO_SERVER_M 1025
6.
7.  #define IP_SERVER_N "127.0.0.1"
8.  #define PO_SERVER_N 1026
9.
10. #define IP_SERVER_C "127.0.0.1"
11. #define PO_SERVER_C 1027
12.
13. ////////////////////////////////////////////////////////////////////
14. ////////////////////// Struttura Pacchetto
15. //////////////////////
16.
17. typedef struct pacchetto_prodotto
18. {
19.     int n_richiesta;
20.
21.     //Prodotto
22.     char nome_prodotto[len_stringa];
23.     float prezzo;
24.
25.     //Negozio
26.     char nome_negozio[len_stringa];
27. }Pacchetto;
28.
29. ////////////////////////////////////////////////////////////////////
30. ////////////////////// LISTA
31. //////////////////////
32.
33. typedef struct nodo_prodotto
34. {
35.     char nome_prodotto[len_stringa];
36.     float prezzo;
37.     struct nodo_prodotto *prev, *next;
38. }Nodo_prodotto;
39.
40. typedef struct nodo_negozio
41. {
42.     char nome_negozio[len_stringa];
```

```
43.     Nodo_prodotto *Lista_prodotti_head, *Lista_prodotti_coda, *Lista_prodotti_work;
44.
45.     struct nodo_negozio *prev, *next;
46. }Nodo_negozio;
47.
48. Nodo_negozio *Lista_negozi_head = NULL, *Lista_negozi_coda = NULL, *Lista_negozi_work = NU
    LL;
49.
50. ////////////////////////////////////////////////////////////////////
51. ////////////////////// FUNCTION LISTA
52. ////////////////////
53.
54. Nodo_negozio* ricerca_negozio(char nome_negozio[len_stringa])
55. {
56.     Nodo_negozio *temp = Lista_negozi_head;
57.
58.     while(temp != NULL)
59.     {
60.         if(strcmp(temp->nome_negozio, nome_negozio) == 0)
61.             return temp;
62.
63.         temp = temp->next;
64.     }
65.
66.     return NULL;
67. }
68.
69. Nodo_prodotto* ricerca_prodotto(char nome_prodotto[len_stringa])
70. {
71.     Nodo_negozio *temp_negozio = Lista_negozi_head;
72.
73.     while (temp_negozio != NULL)
74.     {
75.         Nodo_prodotto *temp_prodotto = temp_negozio->Lista_prodotti_head;
76.         while(temp_prodotto != NULL)
77.         {
78.             if(strcmp(temp_prodotto->nome_prodotto, nome_prodotto) == 0)
79.                 return temp_prodotto;
80.
81.             temp_prodotto = temp_prodotto->next;
82.         }
83.
84.         temp_negozio = temp_negozio->next;
85.     }
86.
87.     return NULL;
88. }
89.
90. int inserimento_negozio(char nome_negozio[len_stringa])
91. {
92.     if(ricerca_negozio(nome_negozio) != NULL)
93.     {
94.         printf("Negozio gia' presente.\n");
95.         return -1;
96.     }
97.
98.     Nodo_negozio *nuovo_nodo;
99.     nuovo_nodo = calloc(1, sizeof(Nodo_negozio));
100.          strcpy(nuovo_nodo->nome_negozio, nome_negozio);
101.          nuovo_nodo->Lista_prodotti_head = NULL;
102.          nuovo_nodo->Lista_prodotti_coda = NULL;
103.          nuovo_nodo->Lista_prodotti_work = NULL;
104.
105.          if (Lista_negozi_head == NULL)///Inserimento primo elemento
106.          {
107.              Lista_negozi_head = nuovo_nodo;
108.              Lista_negozi_coda = nuovo_nodo;
```

```
109.
110.                 Lista_negozi_head->next = NULL;
111.                 Lista_negozi_coda->next = NULL;
112.                 Lista_negozi_work = Lista_negozi_head;///In work ho l'ultimo elemento inser
    ito
113.             }
114.             else
115.             {
116.                 Lista_negozi_work->next = nuovo_nodo;
117.                 Lista_negozi_coda = Lista_negozi_work->next;
118.                 Lista_negozi_coda->prev = Lista_negozi_work;
119.
120.                 Lista_negozi_work = Lista_negozi_coda;
121.                 Lista_negozi_work->next = NULL;
122.             }
123.
124.             return 0;
125.         }
126.
127.         int inserimento_prodotto(char nome_negozio[len_stringa], char nome_prodotto[len_str
    inga], float prezzo)
128.         {
129.             Nodo_negozio *nodo_negozio = ricerca_negozio(nome_negozio);
130.             Nodo_prodotto *nodo_prodotto = ricerca_prodotto(nome_prodotto);
131.
132.             if(nodo_negozio == NULL)
133.             {
134.                 printf("Il negozio non esiste.\n");
135.                 return -1;
136.             }
137.             else if(nodo_prodotto != NULL)
138.             {
139.                 printf("Prodotto gia' presente.\n");
140.                 return -1;
141.             }
142.             else
143.             {
144.                 Nodo_prodotto *nuovo_nodo;
145.                 nuovo_nodo = calloc(1, sizeof(Nodo_prodotto));
146.                 strcpy(nuovo_nodo->nome_prodotto, nome_prodotto);
147.                 nuovo_nodo->prezzo = prezzo;
148.
149.                 if (nodo_negozio-
    >Lista_prodotti_head == NULL)///Inserimento primo elemento
150.                 {
151.                     nodo_negozio->Lista_prodotti_head = nuovo_nodo;
152.                     nodo_negozio->Lista_prodotti_coda = nuovo_nodo;
153.
154.                     nodo_negozio->Lista_prodotti_head->next = NULL;
155.                     nodo_negozio->Lista_prodotti_coda->next = NULL;
156.                     nodo_negozio->Lista_prodotti_work = nodo_negozio-
    >Lista_prodotti_head;///In work ho l'ultimo elemento inserito
157.                 }
158.                 else
159.                 {
160.                     nodo_negozio->Lista_prodotti_work->next = nuovo_nodo;
161.                     nodo_negozio->Lista_prodotti_coda = nodo_negozio->Lista_prodotti_work-
    >next;
162.                     nodo_negozio->Lista_prodotti_coda->prev = nodo_negozio-
    >Lista_prodotti_work;
163.
164.                     nodo_negozio->Lista_prodotti_work = nodo_negozio-
    >Lista_prodotti_coda;
165.                     nodo_negozio->Lista_prodotti_work->next = NULL;
166.                 }
167.             }
168.
```

```
169.            return 0;
170.        }
171.
172.        Nodo_prodotto* inserimento_in_lista(char nome_prodotto[len_stringa], Nodo_prodotto
    *L)
173.        {
174.            Nodo_prodotto *nuovo_nodo, *Lista;
175.            Lista = L;
176.            nuovo_nodo = calloc(1, sizeof(Nodo_prodotto));
177.            strcpy(nuovo_nodo->nome_prodotto, nome_prodotto);
178.            nuovo_nodo->next = NULL;
179.            nuovo_nodo->prev = NULL;
180.
181.            if (Lista == NULL)///Inserimento primo elemento
182.            {
183.                Lista = nuovo_nodo;
184.                Lista->next = NULL;
185.                Lista->prev = NULL;
186.            }
187.            else
188.            {
189.                Lista->next = nuovo_nodo;
190.                Lista->next->next = NULL;
191.                Lista->next->prev = Lista;
192.
193.                Lista = Lista->next;
194.            }
195.
196.            return Lista;
197.        }
198.
199.        int eliminazione_negozio(char nome_negozio[len_stringa])
200.        {
201.            Nodo_negozio *temp, *prev, *curr;
202.
203.            if (Lista_negozi_head != NULL)
204.            {
205.                for (prev = NULL, curr = Lista_negozi_head; curr != NULL && (strcmp(curr-
    >nome_negozio, nome_negozio) != 0); prev = curr, curr = curr->next);
206.                //corr = nodo che sto cercando, prev = precedente al nodo che sto cercando
207.
208.                if (curr == NULL)
209.                {
210.                    printf("Negozio non trovato.\n");
211.                    return -1;
212.                }
213.                else if (prev == NULL)///Eliminazione in testa
214.                {
215.                    temp = Lista_negozi_head->next;
216.                    free(Lista_negozi_head);
217.                    Lista_negozi_head = temp;
218.                    Lista_negozi_head->prev = NULL;
219.                    return 0;
220.                }
221.                else///Eliminazione in mezzo
222.                {
223.                    if (curr->next == NULL)
224.                    {
225.                        Lista_negozi_coda = prev;
226.                        Lista_negozi_coda->next = NULL;
227.                        Lista_negozi_work = Lista_negozi_coda;
228.                        free(curr);
229.                        // Lista_negozi_head = prev;
230.                        // prev->next = NULL;
231.                        // free(curr);
232.                    }
```

```
233.                    else
234.                    {
235.                        prev->next = curr->next;
236.                        curr->next->prev = prev;
237.                        free(curr);
238.                    }
239.                    return 0;
240.                }
241.            }
242.            else
243.                printf("\t\t\tNON CI SONO ELEMENTI DA ELIMINARE\n"); // Lista vuota
244.
245.            return -1;
246.        }
247.
248.        int eliminazione_prodotto(char nome_prodotto[len_stringa])
249.        {
250.            Nodo_negozio *temp_negozio = Lista_negozi_head;
251.
252.            while(temp_negozio != NULL)
253.            {
254.                Nodo_prodotto *temp, *prev, *curr;
255.
256.                if (temp_negozio->Lista_prodotti_head != NULL)
257.                {
258.                    for (prev = NULL, curr = temp_negozio-
    >Lista_prodotti_head; curr != NULL && (strcmp(curr-
    >nome_prodotto, nome_prodotto) != 0); prev = curr, curr = curr->next);
259.                    //corr = nodo che sto cercando, prev = precedente al nodo che sto cerca
    ndo
260.
261.                    if (curr == NULL)
262.                    {
263.                        printf("Prodotto non trovato nel negozio '%s'.\n", temp_negozio-
    >nome_negozio);
264.                    }
265.                    else if (prev == NULL)///Eliminazione in testa
266.                    {
267.                        temp = temp_negozio->Lista_prodotti_head->next;
268.                        free(temp_negozio->Lista_prodotti_head);
269.                        temp_negozio->Lista_prodotti_head = temp;
270.                        temp_negozio->Lista_prodotti_head->prev = NULL;
271.                        return 0;
272.                    }
273.                    else///Eliminazione in mezzo
274.                    {
275.                        if (curr->next == NULL)
276.                        {
277.                            temp_negozio->Lista_prodotti_coda = prev;
278.                            temp_negozio->Lista_prodotti_coda->next = NULL;
279.                            temp_negozio->Lista_prodotti_work = temp_negozio-
    >Lista_prodotti_coda;
280.                            free(curr);
281.                            // temp_negozio->Lista_prodotti_head = prev;
282.                            // prev->next = NULL;
283.                        }
284.                        else
285.                        {
286.                            prev->next = curr->next;
287.                            curr->next->prev = prev;
288.                            free(curr);
289.                        }
290.                        return 0;
291.                    }
292.                }
293.                else
```

```
294.                     printf("Lista prodotti vuota nel negozio '%s'.\n", temp_negozio->nome_negozio); // Lista vuota
295.
296.                 temp_negozio = temp_negozio->next;
297.             }
298.
299.             return -1;
300.         }
301.
302.         ////////////////////////////////////////////////////////////////////
303.         ////////////////////// VISUALIZZAZIONI
304.         //////////////////////
305.
306.         char* elenco_prodotti_del_negozio(char nome_negozio[len_stringa])
307.         {
308.             char *elenco = calloc(len_stringa*len_stringa, sizeof(char));
309.             Nodo_negozio *temp = ricerca_negozio(nome_negozio);
310.
311.             if(temp == NULL)
312.                 return NULL;
313.
314.             Nodo_prodotto *ttemp = temp->Lista_prodotti_head;
315.             while(ttemp != NULL)
316.             {
317.                 strcat(elenco, ttemp->nome_prodotto);
318.                 strcat(elenco, "\n");
319.                 ttemp = ttemp->next;
320.             }
321.
322.             return elenco;
323.         }
324.
325.         char* elenco_negozi()
326.         {
327.             char *elenco = calloc(len_stringa*len_stringa, sizeof(char));
328.             Nodo_negozio *temp = Lista_negozi_head;
329.
330.             if(temp == NULL)
331.                 return NULL;
332.
333.             while(temp != NULL)
334.             {
335.                 strcat(elenco, temp->nome_negozio);
336.                 strcat(elenco, "\n");
337.                 temp = temp->next;
338.             }
339.
340.             return elenco;
341.         }
342.
343.         void visualizza_tutto()
344.         {
345.             Nodo_negozio *temp_negozio = Lista_negozi_head;
346.             while (temp_negozio != NULL)
347.             {
348.                 Nodo_prodotto *temp_prodotto = temp_negozio->Lista_prodotti_head;
349.                 printf("Nome Negozio: %s\n", temp_negozio->nome_negozio);
350.                 while(temp_prodotto != NULL)
351.                 {
352.                     printf("\t\t%s\n", temp_prodotto->nome_prodotto);
353.                     temp_prodotto = temp_prodotto->next;
354.                 }
355.
356.                 temp_negozio = temp_negozio->next;
357.             }
358.         }
359.
```

```
360.        ////////////////////////////////////////////////////////////////////
361.        ////////////////////// INIZIALIZZAZIONE
362.        //////////////////
363.
364.        void inizializzazione()
365.        {
366.            inserimento_negozio("Negozio_dolci");
367.            inserimento_negozio("Negozio_casa");
368.            inserimento_negozio("Negozio_abbigliamento");
369.            inserimento_negozio("Negozio_elettronica");
370.            inserimento_negozio("Negozio_farmacia");
371.
372.            inserimento_prodotto("Negozio_dolci", "Caramelle 1 Kg", 8.99F);
373.            inserimento_prodotto("Negozio_dolci", "Gomme 500g", 5.99F);
374.            inserimento_prodotto("Negozio_dolci", "Cioccolato 250g", 3.99F);
375.            inserimento_prodotto("Negozio_dolci", "Biscotti 1 Kg", 2.99F);
376.            inserimento_prodotto("Negozio_dolci", "Torte 1 Kg", 15.99F);
377.
378.            inserimento_prodotto("Negozio_casa", "Tavolo", 50.99F);
379.            inserimento_prodotto("Negozio_casa", "Sedia", 15.99F);
380.            inserimento_prodotto("Negozio_casa", "Tovaglia", 15.99F);
381.
382.            inserimento_prodotto("Negozio_abbigliamento", "Maglia L", 10.99F);
383.            inserimento_prodotto("Negozio_abbigliamento", "Maglia M", 10.99F);
384.            inserimento_prodotto("Negozio_abbigliamento", "Maglia XS", 10.99F);
385.            inserimento_prodotto("Negozio_abbigliamento", "Maglia XL", 10.99F);
386.            inserimento_prodotto("Negozio_abbigliamento", "Pantalone M", 22.99F);
387.            inserimento_prodotto("Negozio_abbigliamento", "Pantalone L", 22.99F);
388.            inserimento_prodotto("Negozio_abbigliamento", "Pantalone XL", 21.99F);
389.            inserimento_prodotto("Negozio_abbigliamento", "Scarpa 42", 55.99F);
390.            inserimento_prodotto("Negozio_abbigliamento", "Scarpa 46", 55.99F);
391.
392.            inserimento_prodotto("Negozio_elettronica", "HardDisk 1 TB", 49.99F);
393.            inserimento_prodotto("Negozio_elettronica", "Mouse", 39.99F);
394.            inserimento_prodotto("Negozio_elettronica", "Tastiera", 69.99F);
395.            inserimento_prodotto("Negozio_elettronica", "Stampante Laser", 105.99F);
396.            inserimento_prodotto("Negozio_elettronica", "SSD 256 GB", 79.99F);
397.
398.            inserimento_prodotto("Negozio_farmacia", "Tachipirina 1000", 12.99F);
399.            inserimento_prodotto("Negozio_farmacia", "Neambutal 100g", 150.99F);
400.            inserimento_prodotto("Negozio_farmacia", "OKi 15 bustine", 15.99F);
401.        }
```

FILE: ServerM.c

Il file “ServerM.c” permette l’esecuzione del ServerM principale che riceverà pacchetti dai ServerN e ServerC.

```
1.  /*ServerM ha una socket su ServerN e una su ServerC e gestir□ due liste (una per negozi v
    irtuali e una su prodotti)
2.    su cui effettuare ricerca, inserimento ed eliminazione
3.  */
4.  #include"header_fun.h"
5.
6.  int main(int argc, char **argv)
7.  {
8.      inizializzazione();
9.
10.     int sockfd;
11.     sockfd = Socket(AF_INET, SOCK_STREAM, 0);
12.
13.     Setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &(int){ 1 }, sizeof(int));
14.
15.     struct sockaddr_in servaddr;
16.     servaddr.sin_family = AF_INET;
```

```
17.     servaddr.sin_port = htons(PO_SERVER_M);
18.     servaddr.sin_addr.s_addr = inet_addr(IP_SERVER_M); // Accetto tutte le connessioni
19.
20.     Bind(sockfd, (struct sockaddr*)&servaddr, sizeof(servaddr));
21.     Listen(sockfd, 2048); // Crea una coda di attesa per le richieste
22.
23.
24.     ////////////////////////////////////////////////////////////////////
25.     // ISTRUZIONI PER IL SERVER //
26.
27.     fd_set set;
28.     int fd_open[FD_SETSIZE] = {0};
29.     int i, max_fd, n;
30.     struct sockaddr_in cliaddr;
31.
32.     // FD_ZERO(&set);
33.     // FD_SET(sockfd, &set);
34.
35.     max_fd = sockfd;
36.     fd_open[max_fd] = 1;
37.     for( ; ; )
38.     {
39.         FD_ZERO(&set);
40.         ////// Inizializzazione
41.         for(i = sockfd; i <= max_fd; i++)
42.             if(fd_open[i] != 0)
43.                 FD_SET(i, &set);
44.
45.         n = Select(max_fd+1, &set, NULL, NULL, NULL);
46.
47.         ////// Accetto la connessione
48.         if(FD_ISSET(sockfd, &set))
49.         {
50.             n--;
51.
52.             int new_fd = Accept(sockfd, (struct sockaddr*)&cliaddr, &(int){sizeof(cliaddr)
    });
53.
54.             fd_open[new_fd] = 1;
55.
56.             if(max_fd < new_fd)
57.                 max_fd = new_fd;
58.         }
59.
60.         ////// Istruzioni per servire ogni client
61.         i = sockfd;
62.         while(n != 0)
63.         {
64.             i++;
65.
66.             if(fd_open[i] == 0) // Va al prossimo ciclo senza proseguire
67.                 continue;
68.
69.             // SERVO IL CLIENT
70.             if(FD_ISSET(i, &set))
71.             {
72.                 n--;
73.
74.                 /*  Da ServerN
75.                         1 - eliminazione_prodotto
76.                         2 - inserimento_prodotto
77.
78.                         3 - eliminazione_negozio
79.                         4 - inserimento_negozio
80.                     Da ServerC
81.                         5 - ricerca_prodotto
82.                         6 - elenco_prodotti_negozio
```

```
83.
84.                          7 - ricerca_negozio
85.                          8 - elenco_negozi
86.                  */
87.
88.                  Pacchetto pacch_ricevuto;
89.                  char risposta[len_stringa], risposta_2[len_stringa*len_stringa];
90.                  char *temp;
91.                  int n_read;
92.                  n_read = FullRead(i, &pacch_ricevuto, sizeof(pacch_ricevuto));
93.
94.                  switch (pacch_ricevuto.n_richiesta)
95.                  {
96.                  case 1:
97.                      // Eliminazione Prodotto
98.
99.                      if(eliminazione_prodotto(pacch_ricevuto.nome_prodotto) == -1)
100.                         {
101.                             sprintf(risposta, "Eliminazione del prodotto fallita.\n");

102.                             FullWrite(i, &risposta, sizeof(risposta));
103.                         }
104.                         else
105.                         {
106.                             sprintf(risposta, "Eliminazione del prodotto effettuata.\n"
    );
107.                             FullWrite(i, &risposta, sizeof(risposta));
108.                         }
109.
110.                         break;
111.                     case 2:
112.                         // Inserimento Prodotto
113.
114.                         if(inserimento_prodotto(pacch_ricevuto.nome_negozio, pacch_rice
    vuto.nome_prodotto, pacch_ricevuto.prezzo) == -1)
115.                         {
116.                             sprintf(risposta, "Inserimento del prodotto fallito.\n");
117.                             FullWrite(i, &risposta, sizeof(risposta));
118.                         }
119.                         else
120.                         {
121.                             sprintf(risposta, "Inserimento del prodotto effettuato.\n")
    ;
122.                             FullWrite(i, &risposta, sizeof(risposta));
123.                         }
124.
125.                         break;
126.                     case 3:
127.                         // Eliminazione Negozio
128.
129.                         if(eliminazione_negozio(pacch_ricevuto.nome_negozio) == -1)
130.                         {
131.                             sprintf(risposta, "Eliminazione del negozio fallita.\n");
132.                             FullWrite(i, &risposta, sizeof(risposta));
133.                         }
134.                         else
135.                         {
136.                             sprintf(risposta, "Negozio eliminato.\n");
137.                             FullWrite(i, &risposta, sizeof(risposta));
138.                         }
139.
140.                         break;
141.                     case 4:
142.                         // Inserimento Negozio
143.
144.                         if(inserimento_negozio(pacch_ricevuto.nome_negozio) == -1)
145.                         {
```

```
146.                                 sprintf(risposta, "Inserimento del negozio fallito.\n");
147.                                 FullWrite(i, &risposta, sizeof(risposta));
148.                             }
149.                             else
150.                             {
151.                                 sprintf(risposta, "Inserimento del negozio effettuato.\n");

152.                                 FullWrite(i, &risposta, sizeof(risposta));
153.                             }
154.
155.                             break;
156.                         case 5:
157.                             // Ricerca Prodotto
158.
159.                             if(ricerca_prodotto(pacch_ricevuto.nome_prodotto) == NULL)
160.                             {
161.                                 sprintf(risposta_2, "Prodotto non presente.\n");
162.                                 FullWrite(i, &risposta_2, sizeof(risposta_2));
163.                             }
164.                             else
165.                             {
166.                                 sprintf(risposta_2, "Prodotto trovato.\n");
167.                                 FullWrite(i, &risposta_2, sizeof(risposta_2));
168.                             }
169.
170.                             break;
171.                         case 6:
172.                             // Elenco Prodotti Negozio
173.
174.                             temp = elenco_prodotti_del_negozio(pacch_ricevuto.nome_negozio)
    ;
175.                             if(temp == NULL)
176.                             {
177.                                 sprintf(risposta_2, "Negozio non presente.\n");
178.                                 FullWrite(i, &risposta_2, sizeof(risposta_2));
179.                             }
180.                             else
181.                             {
182.                                 strcpy(risposta_2, temp);
183.                                 FullWrite(i, &risposta_2, sizeof(risposta_2));
184.                             }
185.
186.                             break;
187.                         case 7:
188.                             // Ricerca Negozio
189.
190.                             if(ricerca_negozio(pacch_ricevuto.nome_negozio) == NULL)
191.                             {
192.                                 sprintf(risposta_2, "Negozio non presente.\n");
193.                                 FullWrite(i, &risposta_2, sizeof(risposta_2));
194.                             }
195.                             else
196.                             {
197.                                 sprintf(risposta_2, "Negozio trovato.\n");
198.                                 FullWrite(i, &risposta_2, sizeof(risposta_2));
199.                             }
200.
201.                             break;
202.                         case 8:
203.                             // Elenco Negozi
204.
205.                             temp = elenco_negozi();
206.                             if(temp == NULL)
207.                             {
208.                                 sprintf(risposta_2, "Non ci sono negozi.\n");
209.                                 FullWrite(i, &risposta_2, sizeof(risposta_2));
210.                             }
```

```
                            else
                            {
                                strcpy(risposta_2, temp);
                                FullWrite(i, &risposta_2, sizeof(risposta_2));
                            }

                            break;

                        case 9:
                            // Inserisci prodotto nella lista acquisti

                            if(ricerca_prodotto(pacch_ricevuto.nome_prodotto) == NULL)
                            {
                                sprintf(risposta_2, "Prodotto non trovato.\n");
                                FullWrite(i, &risposta_2, sizeof(risposta_2));
                            }
                            else
                            {
                                strcpy(risposta_2, "TRUE\0");
                                FullWrite(i, &risposta_2, sizeof(risposta_2));

                                strcpy(risposta_2, pacch_ricevuto.nome_prodotto);
                                FullWrite(i, &risposta_2, sizeof(risposta_2));
                            }

                            break;

                        default:
                            break;
                        }

                        visualizza_tutto();

                        printf("Chiusa connessione sul Client %d.\n", i);
                        close(i);

                        fd_open[i] = 0;
                        if(max_fd == i)
                        {
                            while(fd_open[i] == 0)
                                i--;
                            max_fd = i;
                            break;
                        }
                    }
                }
            }

            exit(0);
        }
```

FILE: ServerC.c

Il file “ServerC.c” permette l’esecuzione del ServerC che si interfaccia con ClientC e ServerM.

```
/*ServerC avr  una socket su ServerM e una su Client e invia richieste
  al ServerM di ricerca e elenco (di negozi virtuali e prodotti)
*/
#include"header_fun.h"

int main(int argc, char **argv)
{
    int sockfd;
    sockfd = Socket(AF_INET, SOCK_STREAM, 0);

```

```
11.     Setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &(int){ 1 }, sizeof(int));
12.
13.     struct sockaddr_in servaddr;
14.     servaddr.sin_family = AF_INET;
15.     servaddr.sin_port = htons(PO_SERVER_C);
16.     servaddr.sin_addr.s_addr = inet_addr(IP_SERVER_C); // Accetto tutte le connessioni
17.
18.     Bind(sockfd, (struct sockaddr*)&servaddr, sizeof(servaddr));
19.     Listen(sockfd, 2048); // Crea una coda di attesa per le richieste
20.
21.     ////////////////////////////////////////////////////////////////////
22.     // ISTRUZIONI PER IL SERVER //
23.
24.     /*  Da ServerC
25.             5 - ricerca_prodotto
26.             6 - elenco_prodotti_negozio
27.
28.             7 - ricerca_negozio
29.             8 - elenco_negozi
30.     */
31.
32.     fd_set set;
33.     int fd_open[FD_SETSIZE] = {0};
34.     int i, max_fd, n;
35.     struct sockaddr_in cliaddr;
36.
37.     // FD_ZERO(&set);
38.     // FD_SET(sockfd, &set);
39.
40.     max_fd = sockfd;
41.     fd_open[max_fd] = 1;
42.     for( ; ; )
43.     {
44.         FD_ZERO(&set);
45.         ////// Inizializzazione
46.         for(i = sockfd; i <= max_fd; i++)
47.             if(fd_open[i] != 0)
48.                 FD_SET(i, &set);
49.
50.         n = Select(max_fd+1, &set, NULL, NULL, NULL);
51.
52.         ////// Accetto la connessione
53.         if(FD_ISSET(sockfd, &set))
54.         {
55.             n--;
56.
57.             int new_fd = Accept(sockfd, (struct sockaddr*)&cliaddr, &(int){sizeof(cliaddr)
    });
58.
59.             fd_open[new_fd] = 1;
60.
61.             if(max_fd < new_fd)
62.                 max_fd = new_fd;
63.         }
64.
65.         ////// Istruzioni per servire ogni client
66.         i = sockfd;
67.         while(n != 0)
68.         {
69.             i++;
70.
71.             if(fd_open[i] == 0) // Va al prossimo ciclo senza proseguire
72.                 continue;
73.
74.             // SERVO IL CLIENT
75.             if(FD_ISSET(i, &set))
76.             {
```

```
77.                 n--;
78.
79.                 Pacchetto pacch_ricevuto;
80.                 pacch_ricevuto.n_richiesta = -1;
81.                 char risposta[len_stringa*len_stringa];
82.                 int n_read;
83.                 n_read = FullRead(i, &pacch_ricevuto, sizeof(pacch_ricevuto));
84.
85.                 int sockfd_cl = Socket(AF_INET, SOCK_STREAM, 0);
86.
87.                 struct sockaddr_in cliaddr2;
88.                 cliaddr2.sin_family = AF_INET;
89.                 cliaddr2.sin_port = htons(PO_SERVER_M);
90.                 cliaddr2.sin_addr.s_addr = inet_addr(IP_SERVER_M);
91.
92.                 Connect(sockfd_cl, (struct sockaddr*)&cliaddr2, sizeof(cliaddr2));
93.
94.                 FullWrite(sockfd_cl, &pacch_ricevuto, sizeof(pacch_ricevuto));
95.                 FullRead(sockfd_cl, &risposta, sizeof(risposta));
96.
97.                 FullWrite(i, &risposta, sizeof(risposta));
98.
99.                 if(strcmp(risposta, "TRUE") == 0)
100.                        {
101.                            FullRead(sockfd_cl, &risposta, sizeof(risposta));
102.                            FullWrite(i, &risposta, sizeof(risposta));
103.                        }
104.
105.                        if(n_read > 0 || pacch_ricevuto.n_richiesta == 0)
106.                        {
107.                            printf("Chiusa connessione sul Client %d.\n", i);
108.                            close(sockfd_cl);
109.                            close(i);
110.
111.                            fd_open[i] = 0;
112.                            if(max_fd == i)
113.                            {
114.                                while(fd_open[i] == 0)
115.                                    i--;
116.                                max_fd = i;
117.                                break;
118.                            }
119.                        }
120.                    }
121.                }
122.            }
123.
124.            printf("FINE!");
125.            exit(0);
126.        }
```

FILE: ServerN.c

Il file “ServerN.c” permette l’esecuzione del ServerN che si interfaccia con ClientN e ServerM

```
1.  /*ServerN avr<U+FFFD> una socket su ServerM e una su ClientN e invia richieste al ServerM
2.    di eliminazione e inserimento (sulle liste negozi virtuali e prodotti)
3.  */
4.  #include"header_fun.h"
5.
6.  int main(int argc, char **argv)
7.  {
8.      int sockfd;
9.      sockfd = Socket(AF_INET, SOCK_STREAM, 0);
10.
```

```
11.     Setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &(int){ 1 }, sizeof(int));
12.
13.     struct sockaddr_in servaddr;
14.     servaddr.sin_family = AF_INET;
15.     servaddr.sin_port = htons(PO_SERVER_N);
16.     servaddr.sin_addr.s_addr = inet_addr(IP_SERVER_N); // Accetto tutte le connessioni
17.
18.     Bind(sockfd, (struct sockaddr*)&servaddr, sizeof(servaddr));
19.     Listen(sockfd, 2048); // Crea una coda di attesa per le richieste
20.
21.     ////////////////////////////////////////////////////////////////////
22.     // ISTRUZIONI PER IL SERVER //
23.
24.     /*  Da ServerN
25.             1 - eliminazione_prodotto
26.             2 - inserimento_prodotto
27.
28.             3 - eliminazione_negozio
29.             4 - inserimento_negozio
30.     */
31.
32.     fd_set set;
33.     int fd_open[FD_SETSIZE] = {0};
34.     int i, max_fd, n;
35.     struct sockaddr_in cliaddr;
36.
37.     // FD_ZERO(&set);
38.     // FD_SET(sockfd, &set);
39.
40.     max_fd = sockfd;
41.     fd_open[max_fd] = 1;
42.     for( ; ; )
43.     {
44.         FD_ZERO(&set);
45.         ////// Inizializzazione
46.         for(i = sockfd; i <= max_fd; i++)
47.             if(fd_open[i] != 0)
48.                 FD_SET(i, &set);
49.
50.         n = Select(max_fd+1, &set, NULL, NULL, NULL);
51.
52.         ////// Accetto la connessione
53.         if(FD_ISSET(sockfd, &set))
54.         {
55.             n--;
56.
57.             int new_fd = Accept(sockfd, (struct sockaddr*)&cliaddr, &(int){sizeof(cliaddr)
    });
58.
59.             fd_open[new_fd] = 1;
60.
61.             if(max_fd < new_fd)
62.                 max_fd = new_fd;
63.         }
64.
65.         ////// Istruzioni per servire ogni client
66.         i = sockfd;
67.         while(n != 0)
68.         {
69.             i++;
70.
71.             if(fd_open[i] == 0) // Va al prossimo ciclo senza proseguire
72.                 continue;
73.
74.             // SERVO IL CLIENT
75.             if(FD_ISSET(i, &set))
76.             {
```

```
                n--;

                Pacchetto pacch_ricevuto;
                pacch_ricevuto.n_richiesta = -1;
                char risposta[len_stringa];
                int n_read;
                n_read = FullRead(i, &pacch_ricevuto, sizeof(pacch_ricevuto));

                int sockfd_cl = Socket(AF_INET, SOCK_STREAM, 0);

                struct sockaddr_in cliaddr2;
                cliaddr2.sin_family = AF_INET;
                cliaddr2.sin_port = htons(PO_SERVER_M);
                cliaddr2.sin_addr.s_addr = inet_addr(IP_SERVER_M);

                Connect(sockfd_cl, (struct sockaddr*)&cliaddr2, sizeof(cliaddr2));

                FullWrite(sockfd_cl, &pacch_ricevuto, sizeof(pacch_ricevuto));
                FullRead(sockfd_cl, &risposta, sizeof(risposta));

                FullWrite(i, &risposta, sizeof(risposta));

                if(n_read > 0 || pacch_ricevuto.n_richiesta == 0)
                       {
                           printf("Chiusa connessione sul Client %d.\n", i);
                           close(sockfd_cl);
                           close(i);

                           fd_open[i] = 0;
                           if(max_fd == i)
                           {
                               while(fd_open[i] == 0)
                                   i--;
                               max_fd = i;
                               break;
                           }
                       }
                   }
               }
           }

           printf("FINE!");
           exit(0);
       }
```

FILE: clientC.c

```
/*Client avr€ una socket su ServerC e consente di inserire prodotti di diversi
  negozi virtuali in una lista degli acquisti e di visualizzare la lista degli
  acquisti.
*/
#include"header_fun.h"

int main(int argc, char **argv)
{
    //1-CREO LA SOCKET
    int sockfd = Socket(AF_INET, SOCK_STREAM, 0);

    struct sockaddr_in servaddr;
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(PO_SERVER_C);
    servaddr.sin_addr.s_addr = inet_addr(IP_SERVER_C);

    //2-CONNECTION
    Connect(sockfd, (struct sockaddr*)&servaddr, sizeof(servaddr));

```

```
    ////////////////////////////////////////////////////////////////////
    // ISTRUZIONI PER IL CLIENT //

    /*  Da ClientC
            5 - ricerca_prodotto
            6 - elenco_prodotti_negozio

            7 - ricerca_negozio
            8 - elenco_negozi
    */
    Nodo_prodotto *Lista_acquisti = NULL;
    fd_set set;
    int max_fd, scelta = -1;

    while(scelta != 0)
    {
        FD_ZERO(&set); //Reset dei bit
        FD_SET(STDIN_FILENO, &set); //Inserisco in set il fd dello STDIN
        FD_SET(sockfd, &set); //Inserisco in set il fd del socket

        if(sockfd > STDIN_FILENO)
            max_fd = sockfd;
        else
            max_fd = STDIN_FILENO;

        fputs("\nPremi [0] per uscire.\nPremi [5] per ricercare un prodotto.\nPremi [6] per visualizzare l'elenco prodotti.\nPremi [7] per ricercare un negozio.\nPremi [8] per visualizzare l'elenco negozi.\n", stdout);
        fputs("\nPremi [9] per inserire un prodotto nella lista degli acquisti.\nPremi [10] per visualizzare la lista degli acquisti.\n", stdout);

        Select(max_fd+1, &set, NULL, NULL, NULL);

        system("clear");
        fflush(stdin);
        fflush(stdout);

        /*Invio la richiesta*/
        if(FD_ISSET(STDIN_FILENO, &set))
        {
            Pacchetto pacch_inviato;
            Nodo_prodotto *temp;
            char buf[len_stringa];
            fgets(buf, sizeof(buf), stdin);
            scelta = atoi(buf);
            //scanf("%d", &scelta);Nodo_prodotto *Lista_acquisti = NULL;

            switch (scelta)
            {
            case 0:
                pacch_inviato.n_richiesta = scelta;
                FullWrite(sockfd, &pacch_inviato, sizeof(pacch_inviato));
                exit(0);
            case 5:
                // Ricerca Prodotto

                fputs("Inserisci il nome del prodotto da cercare: ", stdout);
                fgets(pacch_inviato.nome_prodotto, sizeof(pacch_inviato.nome_prodotto), stdin);
                //scanf("%s", buf);

                pacch_inviato.nome_prodotto[strlen(pacch_inviato.nome_prodotto) - 1] = '\0';
                pacch_inviato.n_richiesta = scelta;
                FullWrite(sockfd, &pacch_inviato, sizeof(pacch_inviato));

                break;
```

```
82.             case 6:
83.                 // Visualizza elenco prodotti del negozio
84.
85.                 fputs("Inserisci il nome del negozio di cui visualizzare i prodotti: ", st
    dout);
86.                 fgets(pacch_inviato.nome_negozio, sizeof(pacch_inviato.nome_negozio), stdi
    n);
87.
88.                 pacch_inviato.nome_negozio[strlen(pacch_inviato.nome_negozio) - 1] = '\0';

89.                 pacch_inviato.n_richiesta = scelta;
90.                 FullWrite(sockfd, &pacch_inviato, sizeof(pacch_inviato));
91.
92.                 break;
93.             case 7:
94.                 // Ricerca negozio
95.
96.                 fputs("Inserisci il nome del negozio da cercare: ", stdout);
97.                 fgets(pacch_inviato.nome_negozio, sizeof(pacch_inviato.nome_negozio), stdi
    n);
98.
99.                 pacch_inviato.nome_negozio[strlen(pacch_inviato.nome_negozio) - 1] = '\0';

100.                    pacch_inviato.n_richiesta = scelta;
101.                    FullWrite(sockfd, &pacch_inviato, sizeof(pacch_inviato));
102.
103.                    break;
104.                case 8:
105.                    // Visualizza elenco negozi
106.
107.                    pacch_inviato.n_richiesta = scelta;
108.                    FullWrite(sockfd, &pacch_inviato, sizeof(pacch_inviato));
109.
110.                    break;
111.
112.                case 9:
113.                    // Inserisci prodotto nella lista acquisti
114.
115.                    fputs("Inserisci il nome del prodotto da inserire nella lista acqui
    sti: ", stdout);
116.                    fgets(pacch_inviato.nome_prodotto, sizeof(pacch_inviato.nome_prodot
    to), stdin);
117.
118.                    pacch_inviato.nome_prodotto[strlen(pacch_inviato.nome_prodotto) - 1
    ] = '\0';
119.                    pacch_inviato.n_richiesta = scelta;
120.                    FullWrite(sockfd, &pacch_inviato, sizeof(pacch_inviato));
121.
122.                    break;
123.                case 10:
124.                    // Visualizza lista degli acquisti
125.
126.                    temp = Lista_acquisti;
127.                    while(temp != NULL)
128.                    {
129.                        printf("%s\n", temp->nome_prodotto);
130.                        temp = temp->prev;
131.                    }
132.
133.                    break;
134.
135.                default:
136.                    printf("Scelta sbagliata!\n");
137.                    break;
138.                }
139.            }
140.
```

```
141.                /*Ricevo la risposta*/
142.                if(FD_ISSET(sockfd, &set))
143.                {
144.                    char buf[len_stringa*len_stringa];
145.
146.                    FullRead(sockfd, &buf, sizeof(buf));
147.
148.                    if(strcmp(buf, "TRUE") == 0)
149.                    {
150.                        FullRead(sockfd, &buf, sizeof(buf));
151.                        Lista_acquisti =  inserimento_in_lista(buf, Lista_acquisti);
152.                    }
153.                    else
154.                        fputs(buf, stdout);
155.                }
156.            }
157.
158.            exit(0);
159.        }
```

FILE: clientN.c

```
1.  /*ClientN  avr? una socket su ServerN e consente di gestire
2.    i negozi virtuali.
3.  */
4.  #include"header_fun.h"
5.
6.  int main(int argc, char **argv)
7.  {
8.      //1-CREO LA SOCKET
9.      int sockfd = Socket(AF_INET, SOCK_STREAM, 0);
10.
11.     struct sockaddr_in servaddr;
12.     servaddr.sin_family = AF_INET;
13.     servaddr.sin_port = htons(PO_SERVER_N);
14.     servaddr.sin_addr.s_addr = inet_addr(IP_SERVER_N);
15.
16.     //2-CONNECTION
17.     Connect(sockfd, (struct sockaddr*)&servaddr, sizeof(servaddr));
18.
19.     ////////////////////////////////////////////////////////////////////
20.     // ISTRUZIONI PER IL CLIENT //
21.
22.     /*  Da ClientN
23.             1 - eliminazione_prodotto
24.             2 - inserimento_prodotto
25.
26.             3 - eliminazione_negozio
27.             4 - inserimento_negozio
28.     */
29.
30.     fd_set set;
31.     int max_fd, scelta = -1;
32.
33.     while(scelta != 0)
34.     {
35.         FD_ZERO(&set); //Reset dei bit
36.         FD_SET(STDIN_FILENO, &set); //Inserisco in set il fd dello STDIN
37.         FD_SET(sockfd, &set); //Inserisco in set il fd del socket
38.
39.         if(sockfd > STDIN_FILENO)
40.             max_fd = sockfd;
41.         else
42.             max_fd = STDIN_FILENO;
43.
```

```
44.         fputs("\nPremi [0] per uscire.\nPremi [1] per eliminare un prodotto.\nPremi [2] pe
    r inserire un prodotto.\nPremi [3] per eliminare un negozio.\nPremi [4] per inserire un ne
    gozio.\n", stdout);
45.
46.         Select(max_fd+1, &set, NULL, NULL, NULL);
47.
48.         system("clear");
49.         fflush(stdin);
50.         fflush(stdout);
51.
52.         /*Invio la richiesta*/
53.         if(FD_ISSET(STDIN_FILENO, &set))
54.         {
55.             Pacchetto pacch_inviato;
56.             char buf[len_stringa];
57.             fgets(buf, sizeof(buf), stdin);
58.             scelta = atoi(buf);
59.             //scanf("%d", &scelta);
60.
61.             switch (scelta)
62.             {
63.             case 0:
64.                 pacch_inviato.n_richiesta = scelta;
65.                 FullWrite(sockfd, &pacch_inviato, sizeof(pacch_inviato));
66.                 exit(0);
67.             case 1:
68.                 // Elimina un prodotto
69.
70.                 fputs("Inserisci il nome del prodotto da eliminare: ", stdout);
71.                 fgets(pacch_inviato.nome_prodotto, sizeof(pacch_inviato.nome_prodotto), st
    din);
72.                 //scanf("%s", buf);
73.
74.                 pacch_inviato.nome_prodotto[strlen(pacch_inviato.nome_prodotto) - 1] = '\0
    ';
75.                 pacch_inviato.n_richiesta = scelta;
76.                 FullWrite(sockfd, &pacch_inviato, sizeof(pacch_inviato));
77.
78.                 break;
79.             case 2:
80.                 // Inserisci prodotto
81.
82.                 fputs("Inserisci il nome del negozio: ", stdout);
83.                 fgets(pacch_inviato.nome_negozio, sizeof(pacch_inviato.nome_negozio), stdi
    n);
84.                 fputs("Inserisci il nome del prodotto: ", stdout);
85.                 fgets(pacch_inviato.nome_prodotto, sizeof(pacch_inviato.nome_prodotto), st
    din);
86.
87.                 pacch_inviato.nome_negozio[strlen(pacch_inviato.nome_negozio) - 1] = '\0';

88.                 pacch_inviato.nome_prodotto[strlen(pacch_inviato.nome_prodotto) - 1] = '\0
    ';
89.                 pacch_inviato.n_richiesta = scelta;
90.                 FullWrite(sockfd, &pacch_inviato, sizeof(pacch_inviato));
91.
92.                 break;
93.             case 3:
94.                 // Elimina negozio
95.
96.                 fputs("Inserisci il nome del negozio da eliminare: ", stdout);
97.                 fgets(pacch_inviato.nome_negozio, sizeof(pacch_inviato.nome_negozio), stdi
    n);
98.
99.                 pacch_inviato.nome_negozio[strlen(pacch_inviato.nome_negozio) - 1] = '\0';

100.                    pacch_inviato.n_richiesta = scelta;
```

```
101.                         FullWrite(sockfd, &pacch_inviato, sizeof(pacch_inviato));
102.
103.                         break;
104.                     case 4:
105.                         // Inserisci negozio
106.
107.                         fputs("Inserisci il nome del negozio: ", stdout);
108.                         fgets(pacch_inviato.nome_negozio, sizeof(pacch_inviato.nome_negozio
    ), stdin);
109.
110.                         pacch_inviato.nome_negozio[strlen(pacch_inviato.nome_negozio) - 1]
    = '\0';
111.                         pacch_inviato.n_richiesta = scelta;
112.                         FullWrite(sockfd, &pacch_inviato, sizeof(pacch_inviato));
113.
114.                         break;
115.                     default:
116.                         printf("Scelta sbagliata!\n");
117.                         break;
118.                     }
119.                 }
120.
121.                 /*Ricevo la risposta*/
122.                 if(FD_ISSET(sockfd, &set))
123.                 {
124.                     char buf[len_stringa];
125.
126.                     FullRead(sockfd, &buf, sizeof(buf));
127.                     fputs(buf, stdout);
128.                 }
129.             }
130.
131.             exit(0);
132.         }
```