Università degli Studi di Napoli
"Parthenope"
Dipartimento di Scienze e Tecnologie
Corso di Laurea in Informatica

Corso di Reti di Calcolatori

Marketplace

Raffaele Attanasio 0124001695
Dario Musella 0124001799
Raffaele Venuso 0124001754

Anno Accademico 2019/2020

Indice

Capitolo 1

Descrizione del progetto

1.1 Traccia del progetto

La seguente relazione illustra un'implementazione della traccia **Marketplace**. Essa prevede lo sviluppo di un negozio virtuale costituito da più entità: un server *master* mantiene una base di dati cosituita da negozi e prodotti, che interagisce con altri due server che si occuperanno rispettivamente di interfacciarsi con due tipologie di client: acquirenti e negozianti. L'implementazione effettuata si attiene alla traccia, trascurando concetti non richiesti come un sistema di pagamento per gli acquirenti o un meccanismo di gestione delle credenziali basato su tabelle hash. Tali caratteristiche sono state ignorate od implementate in maniera simbolica. A corredo viene riportata la 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 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.

1.2 Definizione sintattica delle entità

Al fine di evitare ambiguità e favorire una maggiore comprensione, da questo punto ci si riferirà alle varie entità utilizzando diversi nomi:

ServerM server *master*, *principale*

ServerC server *cliente*, *acquirente*, *intermedio*[1]

ServerN server *negoziante*, *intermedio*

Client client *acquirente*

ClientN client *negoziante*

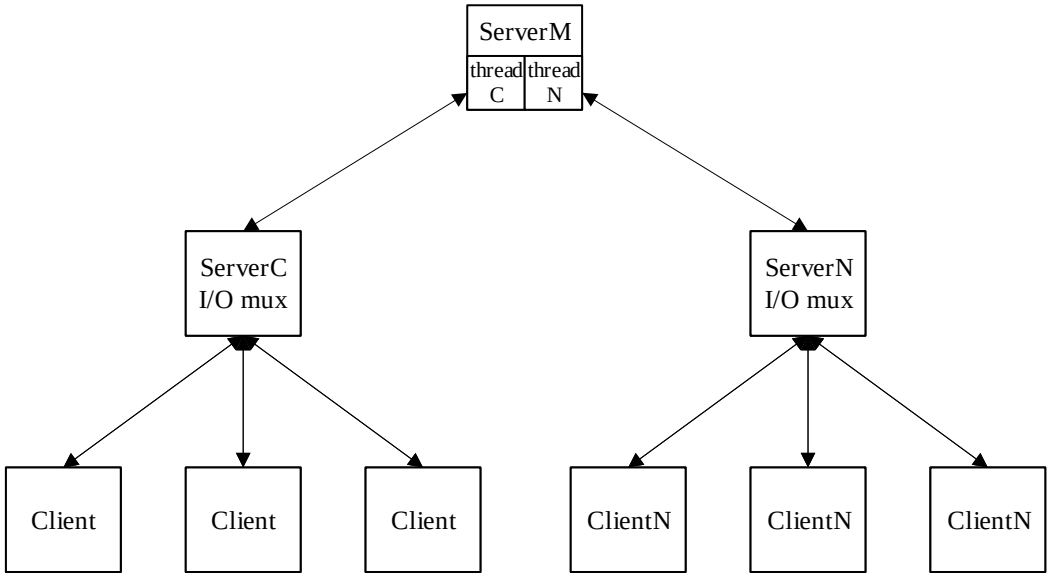
[1]Usato per generalizzare in un contesto i ServerC e ServerN.

Capitolo 2

Descrizione e schemi dell'architettura di rete

La consegna prevede una struttura gerarchica delle entità. L'architettura di rete è stata implementata in maniera **ibrida**, più precisamente:

- Il **serverM** dialoga con gli altri due server eseguendo due **thread** dedicati. Ognuno dialoga con uno dei due server, gestendo esclusivamente le richieste del proprio server di competenza.
- I due **serverC** e **serverN** sono stati sviluppati sul modello dell'**I/O multiplexing**.
- I **client** e **clientN** comunicano con i lori rispettivi server e gestiscono il loro flusso di input/output con delle procedure **bloccanti**.



Capitolo 3

Descrizione e schemi del protocollo applicazione

Di seguito vengono riportati i protocolli usati per la realizzazione delle operazioni svolte dalle varie entità.

3.1 Protocollo per la richiesta di un'operazione

Il protocollo coinvolge tutte le tre entità gerarchicamente differenti. Un client invia un pacchetto contenente il codice della richiesta da eseguire e le informazioni da elaborare. Il pacchetto passa prima dal client al server intermedio, che si occupa di delegare al server principale la richiesta. Una volta arrivato il pacchetto al server master, esso si occupa della lettura del pacchetto e dell'interrogazione della base di dati, quindi invia l'esito della connessione al server intermedio, che a sua volta la inoltra al client.

Va precisato che tale protocollo è applicato ad ambe le tipologie di client, con l'unica differenza data dal sistema di credenziali che precede la prima richiesta di un client negoziante. I dettagli vengono forniti nel paragrafo successivo.

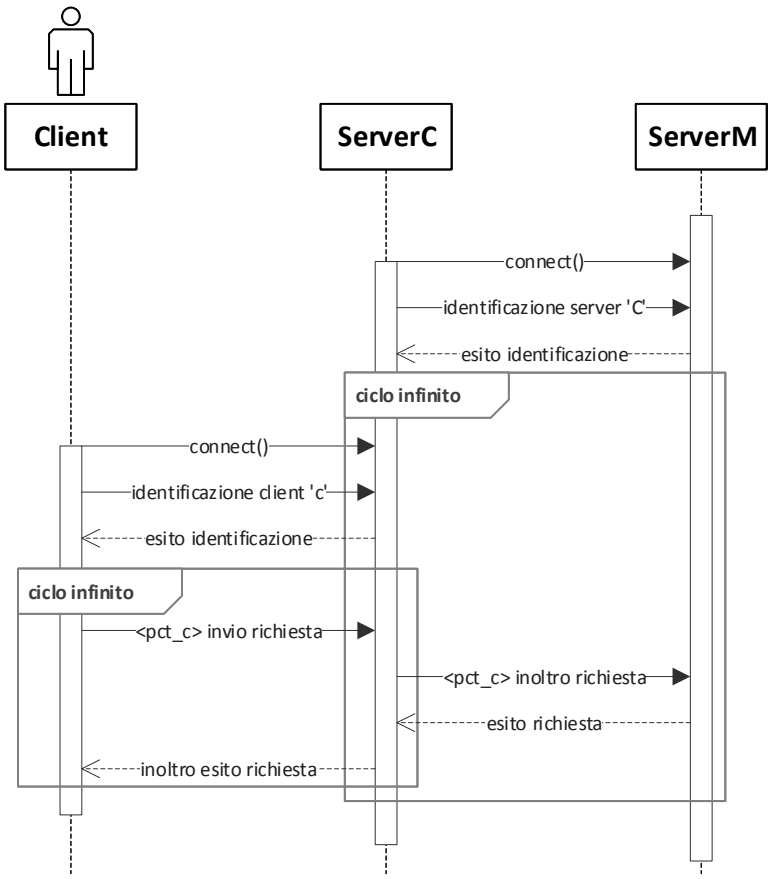


Figura 3.1: Richiesta di un client acquirente

3.2 Protocollo per la gestione delle credenziali

Il protocollo coinvolge le entità negozianti ed il server master. Siccome ogni negozio virtuale è associato ad un utente, è stato implementato un sistema di credenziali che consente la gestione dei negozi di propria competenza. Lo schema prevede, a partire da ClientN l'invio di una richiesta a ServerN contenente il codice inerente all'accesso al proprio profilo, oppure alla registrazione di un nuovo account. Una volta ricevuto il pacchetto e letta la tipologia di richiesta, il ServerN manda un messaggio di conferma al ClientN che successivamente invia un pacchetto di tipo `pct_1` contenente le credenziali. Tale pacchetto passa per il ServerN che lo demanda al server principale. Al termine dell'operazione di verifica da parte del ServerM, invia un messaggio di conferma al server intermedio, che lo gira al client negoziante interessato. Il sistema di gestione delle credenziali consente successivamente di poter effettuare le proprie operazioni, identificandosi ad ogni richiesta.

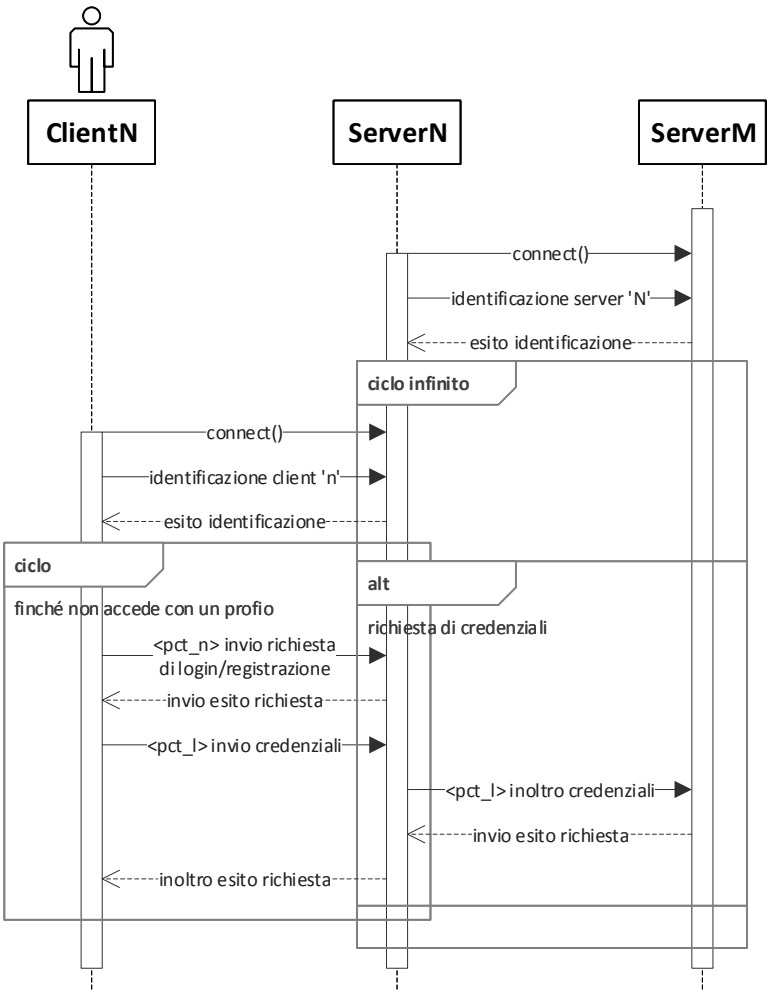


Figura 3.2: Richiesta di login/registrazione di un client negoziante

Capitolo 4

Dettagli implementativi dei client

I client per gli *acquirenti* ed i *negozianti* inviano le richieste al server *master* interpellando rispettivamente con ServerC e ServerN, che si occuperanno di interrogare il server principale. Il dialogo avviene per mezzo di strutture pacchetti e di messaggi di conferma. Le implementazioni delle strutture dati dei pacchetti per le entità acquirenti e negozianti sono fornite qui di seguito:

```
struct pct_c {
        char richiesta;         /* tipologia di richiesta */
        pct_q query;          /* informazioni da elaborare */
}
```

Il pacchetto per il dialogo con le entità negozianti consiste nell'aggiunta di una informazione riguardante l'utente che richiede l'operazione.

```
struct pct_n {
        char richiesta;         /* tipologia di richiesta */
        char user[50];        /* negoziante che richiede */
        pct_q query;          /* informazioni da elaborare */
}
```

La struttura dati pacchetto `pct_q` contiene dei campi che vengono opportunamente riempiti in base alle richieste.

```
struct pct_q {
        char q_neg[50];         /* info negozio */
        char q_prod[50];         /* info prodotto */
}
```

Per le credenziali viene fatto uso di un pacchetto di tipo `pct_l` che contiene rispettivamente username e password del negoziante.

Come affermato prima, i client sono stati sviluppati con delle chiamate **bloccanti** eseguiti in un blocco iterativo, in quanto le operazioni che gli utenti svolgono sono effettuate in maniera sequenziale.

Capitolo 5

Dettagli implementativi dei server

5.1 ServerC e ServerN

I server *cliente* e *negoziante* fungono da tramite per la comunicazione tra i rispettivi clienti acquirenti e negozianti. Una volta connessi al server principale si mettono in attesa di interazione da parte dei client. Il dialogo viene effettuato per mezzo di strutture pacchetto di tipo `pct_c` (per Client e ServerC) e `pct_n` (per ClientN e ServerN) che contengono l'identificativo della richiesta da eseguire ed eventualmente le informazioni che devono essere elaborate.

I due server applicano una **multiplazione** dell'I/O in quanto saranno impegnati nella gestione dei pacchetti che arrivano da n client.

5.2 ServerM

Il server *master* si occupa materialmente della gestione della base di dati. Il processo principale attende la connessione dei due server *cliente* e *negoziante*, quindi genera due **thread** ad-hoc che si occupano della gestione delle richieste. Il meccanismo è strutturato nel seguente modo per il generico thread:

1. Legge la richiesta
2. Accede alla base di dati per elaborare le informazioni
3. Invia i risultati ottenuti

Al fine di annullare le problematiche inerenti alla consistenza dei dati viene applicata una sincronizzazione tra i due thread seguendo l'approccio dei *lettori-scrittori*[2]: quando un negoziante effettua un'operazione di scrittura sul database essa sarà svolta in mutua esclusione. Analogamente le richieste di lettura da parte dei client acquirenti sono eseguite in mutua esclusione.

[2]Un lettore e uno scrittore.

Capitolo 6

Manuale utente

La cartella principale del progetto contiene tre sottocartelle: in `include` sono presenti gli header dei codici delle funzioni contenute nella cartella `lib`, mentre in `src` sono presenti le directory che contengono ognuna i file sorgenti di ogni entità.

6.1 Istruzioni per la compilazione

Per la compilazione è richiesto l'utilizzo del tool **CMake**[3]. Per compilare i codici sorgenti bisogna creare una cartella di compilazione ed entrarci

```
$ mkdir build && cd build
```

Quindi eseguire CMake nella radice del progetto

```
$ cmake ..
```

Infine compilare tutti i codici eseguendo

```
$ make
```

Una volta generati i file eseguibili bisogna inserire il file database `db.txt` all'interno della cartella che contiene gli eseguibili. Il file è contenuto nella radice del progetto.

```
$ cd .. && mv db.txt ./bin
```

6.2 Istruzioni per l'esecuzione

Per eseguire i programmi basta recarsi nella cartella generata, a partire dalla radice

```
$ cd bin
```

Quindi lanciare tutte le entità in sequenza. A partire dal ServerM è possibile lanciarlo richiamando semplicemente l'eseguibile ed aprirlo sulla porta 8000, oppure inserendo come parametro ulteriore `-p` per selezionare arbitrariamente una porta. Se si vuole per esempio avviare il server master sulla porta 5000, il comando da fornire è il seguente

```
$ ./serverM -p 5000
```

[3]Versione minima: 3.10

Successivamente bisogna lanciare i due server intermedi dei clienti e negozianti. In questi casi è necessario esplicitare l'indirizzo di connessione del server principale mediante il parametro `-i`. Facoltativamente è possibile scegliere sia la porta dove il server intermedio deve connettersi per dialogare con il ServerM sia la porta da dove potranno connettersi i propri client, fornendo rispettivamente i parametri `-m` e `-p`. Seguendo l'esempio precedente e simulando una connessione in locale, per eseguire i server intermedi esplicitiamo l'indirizzo di connessione del server principale, assieme alla sua porta. Inoltre si rendono accessibili i client negozianti ed acquirenti rispettivamente sulle porte 5001 e 5002.

```
$ ./serverN -i localhost -m 5000 -p 5001
$ ./serverC -i localhost -m 5000 -p 5002
```

Una volta eseguiti i server è possibile eseguire i client. Accanto al nome dell'eseguibile è necessario fornire l'indirizzo di connessione del proprio server di competenza, opzionalmente anche la porta. Completando l'esempio riportato, si vogliono eseguire un client negoziante ed acquirente.

```
$ ./clientN -i localhost -p 5001
$ ./client -i localhost -p 5002
```