Università degli Studi di Messina

Scienze Informatiche

Antonio Edoardo Ciliberto
Emanuele Russo

Progetto LabReSiD23

21 Giugno 2024

Indice

Stato dell'arte

Introduzione al settore della gestione dei ricambi

La gestione dei ricambi è cruciale per diverse industrie, inclusi settori come l'automotive, l'industria manifatturiera, e il retail. Le soluzioni di gestione dei ricambi consentono di tenere traccia degli inventari, delle vendite e delle operazioni di manutenzione, garantendo che i componenti necessari siano sempre disponibili quando richiesti.

Soluzioni tradizionali

In passato, la gestione dei ricambi era spesso effettuata manualmente o utilizzando software desktop specializzati. Questi sistemi richiedevano l'installazione su ogni macchina client e non sempre permettevano un accesso centralizzato o aggiornamenti in tempo reale dei dati.

Sistemi moderni di gestione dei ricambi

Negli ultimi anni, sono emersi sistemi più avanzati che sfruttano le tecnologie web e le applicazioni client-server per migliorare l'efficienza e l'accessibilità della gestione dei ricambi:

- Applicazioni Web
- Tecnologie Cloud
- Sistemi Client-Server

Descrizione del problema

Descrizione

Realizzare in C un sistema multithreading client/server per la gestione da remoto di un magazzino ricambi secondo il paradigma CRUD usando SQLite.

Implementazione

Architettura dell'applicazione

Il lato client dell'applicazione è stato progettato per gestire interazioni utente e comunicazioni con il server mediante l'uso di socket TCP/IP. L'architettura è basata su un modello di menù interattivo che permette agli utenti di eseguire le operazioni CRUD (Create, Read, Update, Delete).

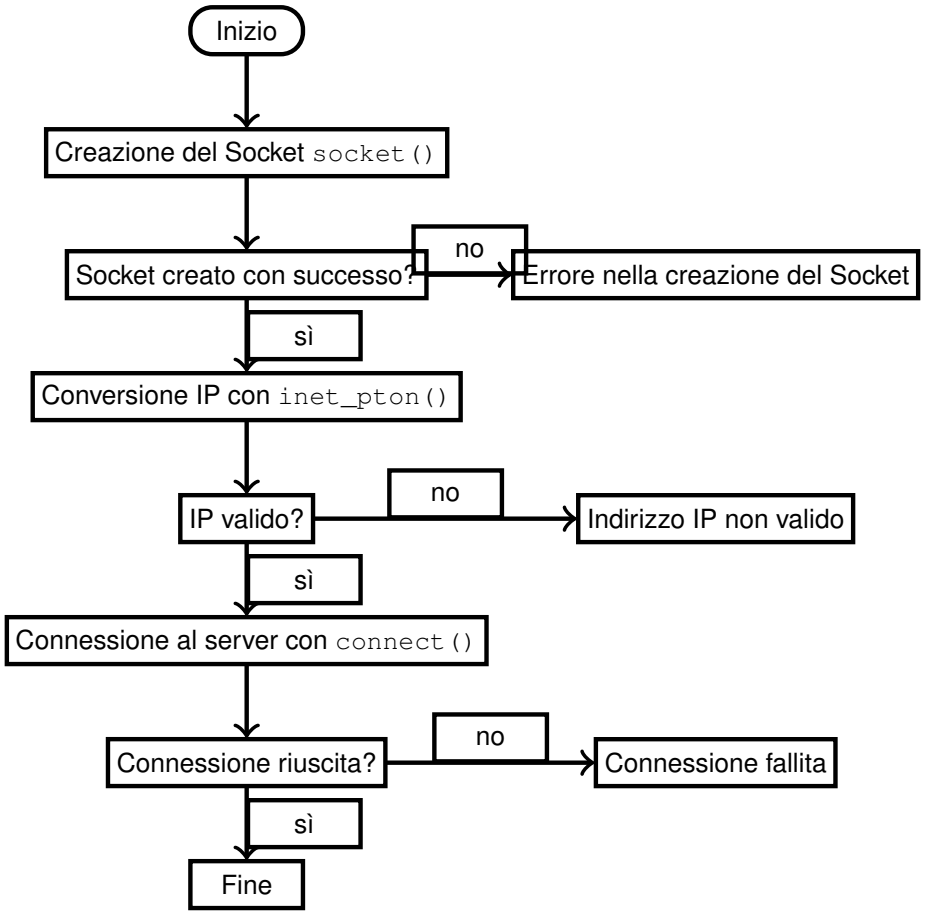
Linguaggi e tecnologie utilizzate

Il client è stato sviluppato in linguaggio C utilizzando le seguenti librerie standard:

- **<stdio.h>**
- **<stdlib.h>**
- **<string.h>**
- **<unistd.h>**
- **<termios.h>**
- **<arpa/inet.h>**

Connessione al server

- Il client crea un socket TCP/IP tramite la funzione `socket()` per stabilire una connessione con il server.
- Utilizza `inet_pton()` per convertire l'indirizzo IP "127.0.0.1" nel formato richiesto dalla struttura `sockaddr_in`.
- Utilizza `connect()` per connettersi alla porta specificata (8080).



Modalità di accesso

Il client gestisce due modalità di accesso: 'Amministratore' e 'Utente'.

- Amministratore: deve inserire una password per accedere alle funzionalità avanzate.
- Utente: ha la possibilità di visualizzare i ricambi disponibili ed acquistarli.

Menù utente e amministratore

Il client presenta un menù interattivo basato sulla modalità di accesso selezionata (Amministratore o Utente).

- Amministratore:

```
Menu Amministratore:
1. Aggiungi un nuovo ricambio
2. Visualizza i ricambi
3. Aggiorna un ricambio
4. Elimina un ricambio
5. Esci
Scelta:
```

.

- Utente:

```
Menu Utente:
1. Visualizza i ricambi
2. Acquista un ricambio
3. Esci
Scelta:
```

.

Implementazione delle operazioni sui ricambi

Il client implementa le seguenti operazioni sui ricambi:

- **Funzione per aggiungere un ricambio**
- **Funzione per visualizzare i ricambi**
- **Funzione per aggiornare i ricambi**
- **Funzione per eliminare un ricambio**
- **Funzione per acquistare un ricambio**

Comunicazione con il server

La comunicazione con il server avviene tramite l'invio di query SQL formate dinamicamente in base all'operazione richiesta dall'utente. Le query vengono inviate utilizzando **'send()'** e le risposte del server vengono lette tramite **'read()'**. Questo processo gestisce sia le richieste di operazioni CRUD che la conferma degli acquisti.

```
1 // Esempio di invio di una query SQL al server
2 send(sock, query, strlen(query), 0);
3 // Esempio di lettura della risposta dal server
4 char response[BUFFER_SIZE];
5 int valread = read(sock, response, BUFFER_SIZE);
6 printf("%s\n", response);
7
```

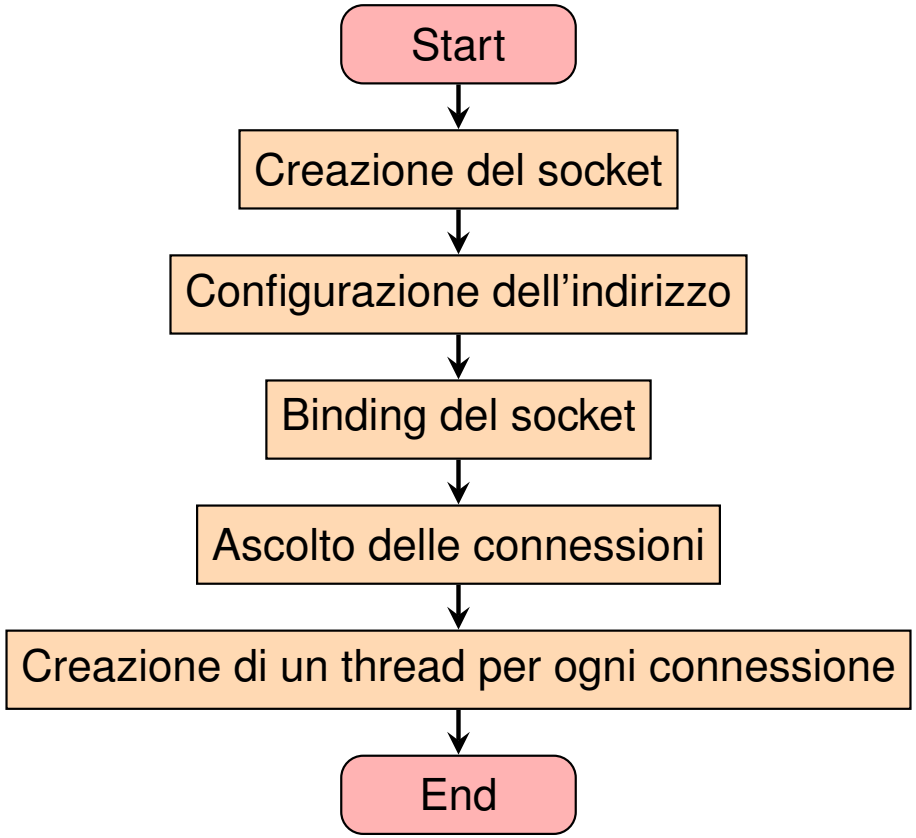
Tecnologie e strumenti utilizzati

Il server è stato sviluppato in linguaggio C utilizzando le seguenti librerie:

- **<stdio.h>**
- **<stdlib.h>**
- **<string.h>**
- **<unistd.h>**
- **<pthread.h>**
- **<sqlite3.h>**
- **<netinet/in.h>**
- **<arpa/inet.h>**

Creazione del server

Il server utilizza un socket TCP/IP per ascoltare le connessioni sulla porta 8080. Esso è progettato per gestire fino a 100 connessioni simultanee. Quando un nuovo client si connette, il server accetta la connessione e crea un nuovo thread per gestire la comunicazione con il client.

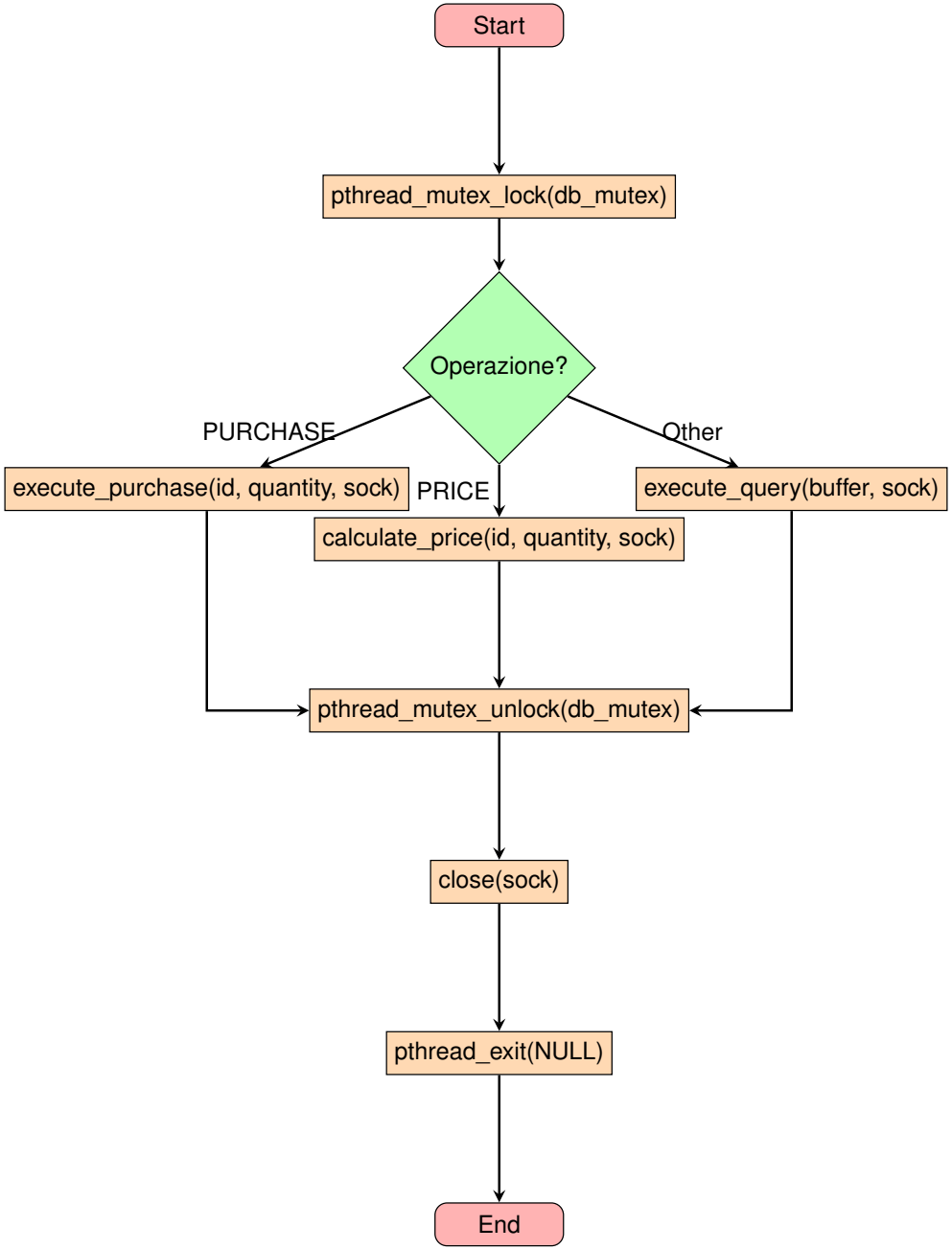


Sincronizzazione e mutua esclusione

Per garantire la consistenza dei dati nel database, il server utilizza un mutex (`pthread_mutex_t db_mutex`). Ogni volta che un thread accede al database per eseguire una query, il mutex viene bloccato per assicurare che nessun altro thread possa accedere contemporaneamente al database.

```
1  sqlite3 *db;
2  pthread_mutex_t db_mutex; // Mutex per la mutua esclusione sul database
3  ...
4  pthread_mutex_lock(&db_mutex); // Lock per la mutua esclusione sul database
5  if (strncmp(buffer, "PURCHASE", 8) == 0) {
6      int id, quantity;
7      sscanf(buffer, "PURCHASE %d %d", &id, &quantity);
8      execute_purchase(id, quantity, sock);
9  } else if (strncmp(buffer, "PRICE", 5) == 0) {
10     int id, quantity;
11     sscanf(buffer, "PRICE %d %d", &id, &quantity);
12     calculate_price(id, quantity, sock);
13 } else {
14     execute_query(buffer, sock);
15 }
16 pthread_mutex_unlock(&db_mutex); // Unlock per la mutua esclusione sul database
17 close(sock);
18 pthread_exit(NULL);
19
```

Implementazione server



Gestione delle richieste dei client

Il server è in grado di gestire diversi tipi di richieste dai client:

- **Acquisto di un ricambio**: Identificato dalla stringa **"PURCHASE"**
- **Calcolo del prezzo**: Identificato dalla stringa **"PRICE"**
- **Esecuzione di query SQL**: Gestisce query SQL generiche, con particolare attenzione alle query di tipo '**SELECT**'

Esecuzione di query SQL

Le query SQL vengono eseguite in due modalità principali:

- **Query di selezione**: Viene utilizzata la funzione **'sqlite3_prepare_v2()'** per preparare la query, **'sqlite3_step()'** per iterare sui risultati e **'sqlite3_column_text()'** per ottenere i valori delle colonne. I risultati vengono inviati al client formattati come testo.
- **Query di modifica**: Viene utilizzata **'sqlite3_exec()'** per eseguire query di inserimento, aggiornamento o cancellazione. In caso di errore, viene inviata una risposta appropriata al client.

Implementazioni delle funzioni

- **Funzione di gestione delle richieste del client**: Riceve i dati dal client, analizza il tipo di richiesta e chiama le funzioni appropriate (**'execute_purchase'**, **'calculate_price'** o **'execute_query'**). Ogni accesso al database è protetto da un lock sul mutex **'db_mutex'**.
- **Funzione del processo d'acquisto**: Questa funzione gestisce il processo di acquisto di un ricambio, recupera la quantità e il prezzo corrente del ricambio dal database. Inoltre verifica la disponibilità della quantità richiesta, aggiorna la quantità nel database e calcola e invia il prezzo totale al client.
- **Funzione del calcolo per il prezzo finale**: La funzione calcola il prezzo totale per una determinata quantità di ricambi.
- **Funzione della gestione delle query**: Gestisce le query SQL generiche. Se la query è di tipo **'SELECT'**, prepara ed esegue la query, iterando sui risultati e inviandoli al client.

Database SQLite

SQLite è una libreria C che fornisce un database SQL leggero e full-featured. A differenza dei tradizionali sistemi di gestione di database relazionali (RDBMS), SQLite è incorporato all'interno dell'applicazione che utilizza il database. Nel contesto della gestione dei ricambi il database, chiamato **'magazzino.db'**, viene utilizzato per memorizzare i dati relativi ad essi.

Creazione database

La tabella chiamata **'Ricambi'** presenta i seguenti campi:

- **ID**: Chiave primaria.
- **Nome**: Nome del ricambio.
- **Descrizione**: Descrizione del ricambio.
- **Quantità**: Quantità disponibile in magazzino.
- **Prezzo**: Prezzo unitario del ricambio.

```
1  ...
2  int rc = sqlite3_open("magazzino.db", &db);
3  ...
4  const char *sql = "CREATE TABLE IF NOT EXISTS Ricambi("
5      "ID INTEGER PRIMARY KEY AUTOINCREMENT, "
6      "Nome TEXT, "
7      "Descrizione TEXT, "
8      "Quantit  INTEGER, "
9      "Prezzo REAL);";
10
```

Risultati sperimentali

Risultati sperimentali

I risultati ottenuti dalla realizzazione del sistema multithreading client/server per la gestione da remoto di un magazzino ricambi, confermano uno degli obiettivi principali del progetto, ovvero, quello di garantire la gestione concorrente delle richieste client tramite un approccio multithreading.

Test

- Supponendo che il Client A abbia acquisito il mutex per primo, e prova ad acquistare 2 pezzi di un prodotto.

```
ID = 14
Nome = Pinza freno assale anteriore sinistro
Descrizione = Pinza freno assale anteriore sinistro per Bmw: 3 Touring, 2 Coupe
Quantità = 3 pz
Prezzo = 26.63 €

ID = 44
Nome = Pneumatici
Descrizione = Pneumatico invernale da 17 pollici
Quantità = 10 pz
Prezzo = 125.0 €


------------------------------------------------------------

Inserisci l'ID del ricambio che desideri acquistare (0 per tornare indietro): 14
Inserisci la quantita che desideri acquistare: 2
Il prezzo totale per 2 pezzi è: 53.26 euro.
Vuoi procedere con l'acquisto? (S/N): s
Acquisto completato. Il prezzo totale è: 53.26 euro.


Menu Utente:
1. Visualizza i ricambi
2. Acquista un ricambio
3. Esci
Scelta:
```

Test

- Successivamente, il Client B, cerca di accedere al database quando ancora il Client A non ha completato l'acquisto e visualizza la presenza di 3 pezzi disponibili e procede anche lui con l'acquisto di 2 pezzi contemporaneamente al Client A.

```
ID = 14
Nome = Pinza freno assale anteriore sinistro
Descrizione = Pinza freno assale anteriore sinistro per Bmw: 3 Touring, 2 Coupe
Quantità = 3 pz
Prezzo = 26.63 €

ID = 44
Nome = Pneumatici
Descrizione = Pneumatico invernale da 17 pollici
Quantità = 10 pz
Prezzo = 125.0 €


------------------------------------------------------------

Inserisci l'ID del ricambio che desideri acquistare (0 per tornare indietro): 14
Inserisci la quantita che desideri acquistare: 2
Il prezzo totale per 2 pezzi è: 53.26 euro.
Vuoi procedere con l'acquisto? (S/N): s
Quantità insufficiente in magazzino.


Menu Utente:
1. Visualizza i ricambi
2. Acquista un ricambio
3. Esci
Scelta:
```

Risultato test

Il test ha confermato la mutua esclusione quindi la possibilità di accedere al database evitando race condition.
Inoltre i log del server confermano che entrambe le operazioni sono state eseguite in modo sicuro e che il database riflette correttamente lo stato finale delle scorte.

Risultati sperimentali

I test di esecuzione dell'applicazione confermano che il sistema implementato gestisce correttamente le richieste concorrenti utilizzando un modello multithreading e meccanismi di mutua esclusione. Questo garantisce la coerenza dei dati nel database SQLite e l'affidabilità delle operazioni CRUD.

Conclusioni e sviluppi futuri

Conclusioni e sviluppi futuri

Il progetto di gestione da remoto di un magazzino di ricambi ha dimostrato l'efficacia di un sistema client-server basato su socket TCP/IP per la gestione centralizzata e l'interazione remota con un database. Garantendo, soprattutto, la gestione concorrente delle risorse tramite il multithreading. Il progetto, inoltre, pone solide basi per delle future implementazioni, come:

- Implementazione di una GUI più intuitiva.
- Implementazione di Login/Registrazione.
- Implementazione di una sezione 'carrello' per acquisti multipli.
- Servizio di resi e rimborsi dei prodotti acquistati.