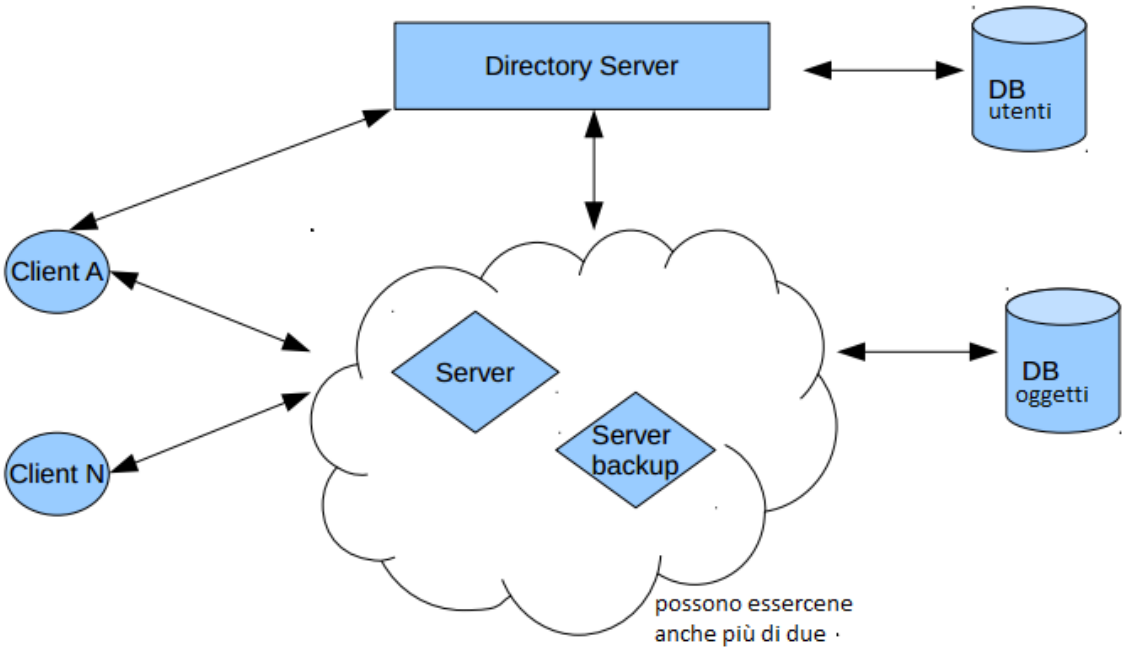
Gestore aste online

Obbiettivo

L'obbiettivo del progetto è lo sviluppo di un gestore di aste, implementato usando un directory server, uno o più server e dei clients che interagiscono attraverso il protocollo TCP.



Il compito del directory server è quello di gestire la registrazione di nuovi utenti, o controllare il corretto login degli stessi, basandosi su un database. Il directory server gestisce anche tutti i servers che vi si connettono, passando poi agli utenti che effettuano il login, l'indirizzo IP e la porta a cui il client si dovrà connettere per fare offerte o mettere in vendita degli oggetti. Per memorizzare le credenziali degli utenti utilizza un database MySQL e tutte le query vengono gestite attraverso le API C di MySQL.

I servers si connettono al directory server, informandolo della porta su cui attenderanno le connessioni dei client. Anche I servers utilizzano un database MySQL per contenere e gestire tutte le informazioni relative agli oggetti e alle offerte. I clients si connettono all'indirizzo del directory server, che poi li informerà dell'indirizzo IP e della porta dell'ultimo server connesso.

Directory server

Il directory server è implementato tramite I/O multiplexing, in modo da accorgersi nel caso di caduta dei servers o dei clients che vi sono connessi. In questo caso il multiplexing è adatto, dato che un client deve solo effettuare il login, procedura che impiega poco tempo a concludersi.

Clients e servers sono divisi dal directory server in due liste, nelle quali gli ultimi clients (inteso in senso generale) connessi vengono aggiunti in testa. Alla connessione, client e server, vengono aggiunti ad una lista temporanea, fino a quando non inviano informazioni sulla natura del servizio offerto: un nuovo server che si collega al directory server si riconoscerà perché come prima stringa invierà la dicitura “new server” mentre un nuovo client invierà la stringa “new client”. Quando il directory server riconosce che tipo di servizio offre il client connesso, il nodo temporaneo sarà spostato nella lista server_list o client_list (nel caso si tratti di un server o un client rispettivamente).

La struttura della lista è comune per nodo temporaneo, nodo server e nodo client:

```
TYPEDEF STRUCT LIST {
        INT SERVER_FD;
        STRUCT SOCKADDR_IN SVRADDR;
        INT LEN;
        INT PORTNUMBER; /* ONLY SET IF IT IS A SERVER */
        STRUCT TIMEVAL ADDTIME;
        STRUCT CLIENT_INFO *ADD_INFO; /* ONLY SET IF IT IS A CLIENT */
        STRUCT LIST *NEXT;
} LIST_NODE;
```

Il primo parametro indica il file descriptor del client, il secondo è la struttura sockaddr_in(contenente le informazioni relative a indirzzo IP e porta), len è un parametro di tipo socklen_t (non viene usato in realtà), se è un server il parametro portNumber sarà diverso da 0, il parametro addTime indica l'orario in cui è stato creato il nodo (anche questo non utilizzato), next è il puntatore al prossimo nodo della lista, mentre il puntatore add_info è una struttura di tipo

```
TYPEDEF STRUCT CLIENT_INFO {
        INT LOGGED;
        CHAR USERNAME[100];
        CHAR PASSWORD[100];
} CLIENT_INFO;
```

che viene allocate solo se si tratta di un client (altrimenti punta a NULL).

Il nodo creato viene spostato nella lista server_list o client_list senza distruggerlo e ricrealo, e vengono settati i parametri relativi (portNumber se è un server, tutti i parametri della struttura add_info ed il relative puntatore se è un client).

Server

Un server si comporta prima come un client nei confronti di un directory server, connettendosi a quest'ultimo ed informandolo della porta su cui rimane in ascolto (l'indirizzo IP del server è noto al directory server essendo lo stesso che vede quando accetta la connessione) e poi come un server nei confronti dei client che si connetteranno per richiedere informazioni sugli oggetti in vendita. Il server è di tipo concorrente, ogni processo figlio gestisce un singolo client. Questo perché le operazioni possibili potrebbero richiedere molto tempo (rispetto al semplice controllo del login, ma sempre in termini di millisecondi). Come prima cosa, il server invia al client che si connette un menù con le varie operazioni disponibili. Farà poi dei controlli sulle stringhe, attraverso le funzioni strstr o strcmp (la prima controlla se una stringa ne contiene un altra, la seconda l'ho utilizzata per controllare se due stringhe fossero uguali o meno), per decidere le operazioni da eseguire. Ho scelto di utilizzare un buffer di grandezza 8192 byte, grande a sufficienza per contenere le stringhe più lunghe generabili dai processi server e client. Le operazioni sul database vengono effettuate tramite l'API C MySQL usando delle semplici SELECT, UPDATE ed INSERT.

Client

Anche il client è implementato usando l'I/O multiplexing. Per prima cosa si collega al directory server: se questo ha dei nodi nella lista server_list, il client si collegherà al primo di essi, ricevendo indirizzo ip e numero della porta dal directory server (questi seguono la costante WELCOME_CLIENT_LOGGED e vengono parsificati poi da una funzione). Se non vi sono server nella lista, il client riprova tre volte a chiedere l'indirizzo ip e la porta di un server a cui collegarsi (con un'attesa di 5 secondi tra ogni richiesta) e, nel caso la lista sia vuota, esce normalmente, notificando all'utente di riprovare in seguito. Il client mantiene la connessione attiva con il directory server anche dopo essersi connesso ad un server, in modo da poter ricevere nuovi dati dal primo se necessario. Una volta connesso, il client costruisce la stringa di risposta prendendo l'input dell'utente concatenato a mini stringhe ben definite, in modo da essere riconosciute dal directory server e dal server. Con dei piccoli cambiamenti, client e server sarebbero in grado di continuare lo scambio di dati nel caso di caduta del server come se nulla fosse successo (in realtà questo non succede, perché il server non mantiene informazioni sui client e vede qualsiasi richiesta come ricevuta da un nuovo client, quindi inviando nuovamente il menù principale alla riconnessione, e forzando la costruzione della stringa di riposta del client). In caso di caduta del directory server, se il client è connesso ad un server, continuerà a lavorare

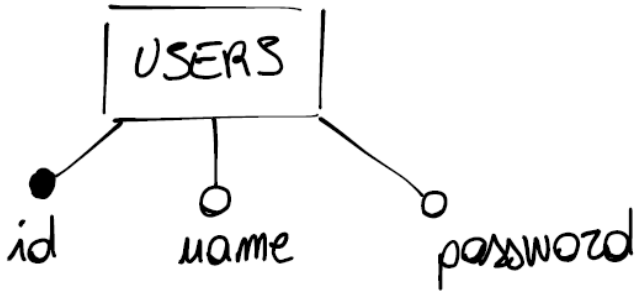
normalmente; in caso di caduta anche di questo, l'utente verrà avvisato, ed il client terminerà l'esecuzione. Quando un utente vuole acquistare un oggetto, devo prima fare una ricerca, o per nome, o visualizzando gli ultimi 10 oggetti messi in vendita; può quindi selezionare l'oggetto su cui vuole fare un'offerta inserendo il suo id, ed, infine, inserire il prezzo che vuole offrire. Un utente può anche visualizzare gli oggetti su cui ha fatto delle offerte, gli oggetti che ha vinto, e gli oggetti in vendita.

Struttura dei file e del database

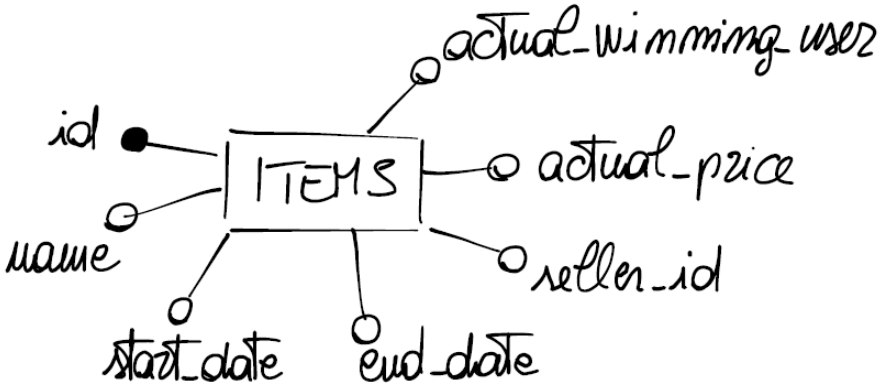
Il file config.h viene utilizzato per facilitare la formulazione di richieste e risposte, semplificando anche una possibile traduzione del tutto (mi piace scrivere i programmi in inglese per un possibile riutilizzo futuro). I file list.c e list.h contengono funzioni, primitive e strutture relative alle liste utilizzate da directory server e server. Mysql.c e mysql.h contengono funzioni, primitive, variabili e strutture necessarie per le queries SQL. I file my_functions.c e my_functions.h contengono funzioni e primitive necessarie per parsificare le stringhe di tutti i processi (svolgono uno dei lavori più importanti, date che tutti i processi usano molto la parsificazione di stringhe per definire che dati stanno ricevendo ed essere in grado di rispondere nel modo corretto).

Ho scelto una struttura del database molto semplice: tre tabelle in tutto

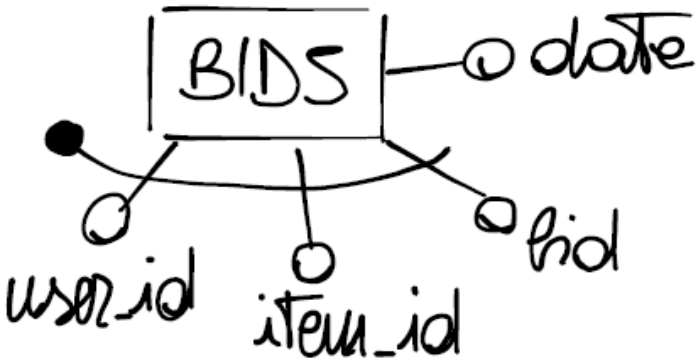
1. users, contenente id, username e password di tutti gli utenti iscritti



2. items, contentente id, nome, data di inizio asta, data di fine asta, id dell'utente venditore, prezzo attuale e id del vincitore attuale



3. bids, contenente id dell'utente, id dell'oggetto, prezzo puntato e data di quando è avvenuta l'offerta



La tabella bids viene utilizzata per poter mostrare quali sono gli oggetti su cui si è fatta un'offerta se non si è più i vincitori attualmente. Quando un utente fa un'offerta su un oggetto, si controlla se l'oggetto con quell'id esiste, se l'utente che fa l'offerta è lo stesso che ha messo in vendita l'oggetto (non può fare l'offerta), se è già il vincitore con l'offerta attuale (l'offerta rimane invariata), se l'offerta supera il valore presente nel database (il massimo attuale) ed in questo caso viene aggiornato il massimo e viene aggiunta una tupla nella tabella bids; in ogni caso viene presentato nuovamente il menù principale.

Di default il database utilizzato è quello relativo alla mia matricola, 10035843, e con password 10035843. Le tabelle, se non esistono già, vengono create all'avvio (la tabella users alla'avvio del directory server, le tabelle items e bids all'avvio di un server). È necessario creare almeno due utenti per testare tutte le funzionalità dei programmi, dato che non è possibile fare un'offerta su un oggetto se si è anche I venditori di tale oggetto.

Un problema che potrebbe verificarsi è l'impossibilità di un client di connettersi ad un qualsiasi server, se directory server e server vengono lanciati sulla stessa macchina: questo perché un server che si connette ad un directory server indicando come hostname localhost, viene memorizzato dal directory server come avente indirizzo 127.0.0.1; il client, se proveniente da un'altra macchina, ricevendo questo indirizzo, proverà a connettersi ad un processo sulla porta specificata all'indirizzo 127.0.0.1 (la stessa macchina del client) e probabilmente riceverà un connection refused.

All'interno della cartella del progetto, 10-GestoreAsteOnline, si trova un file readme.txt che riepiloga I passi da seguire per registrare un nuovo utente, loggarsi come utenti, mettere in vendita un oggetto o fare un'offerta.