Modulo extra: microSQL (μSQL)

Laboratorio di Sistemi Operativi I
Anno Accademico 2005-2006

Francesco Pedullà
(Tecnologie Informatiche)

Massimo Verola
(Informatica)

Il progetto

Questo modulo descrive il progetto da svolgere per l'esame di Laboratorio di Sistemi Operativi I.

I progetto richiede lo sviluppo di quattro componenti software, di cui tre necessarie alla realizzazione di un semplice DBMS (Data Base Management System) ed un'applicazione di test che lo utilizza:

- **DB Server**: demone che gestisce l'accesso al DB da parte delle applicazioni client
- **DB Admin**: interfaccia command-line per il DB administrator dedicata al monitoraggio e controllo del DB Server
- **DB Lib**: libreria necessaria per la programmazione di applicazioni client.
- **Hotel**: applicazione client che utilizza DB Lib per implementare un sistema di gestione delle camere di un albergo.

Il Database (1)

- Un DB e' essenzialmente un insieme di tabelle, le quali contengono una lista di record (ovvero le righe della tabella), i quali sono composti da un certo numero di campi informativi (ovvero le colonne della tabella)
- Il DB del progetto deve poter contenere un numero potenzialmente illimitato di tabelle
- DB differenti devono essere distinti da nomi diversi e possono essere implementati come directory
- Le tabelle possono essere implementate come file
- Ogni tabella deve poter contenere un numero potenzialmente illimitato di record

Il Database (2)

- Tutti i record di una stessa tabella hanno lo stesso numero di campi
- L'unico tipo di campo da supportare e' la stringa, che puo' essere anche di lunghezza fissa predefinita (ad es. 20 caratteri)
- I record devono poter contenere un numero potenzialmente illimitato di campi
- I campi dei record possono essere implementati in formato CSV (Comma-Separated Values), con un carattere di newline per separare 2 record (cioe' ogni record su una linea del file testo)
- Si raccomanda di usare un formato leggibile dei file per semplificare la verifica ed il debugging

DB Server (1)

Il programma DB Server, il cui file eseguibile prendera' il nome *dbserv*, deve (vedi le figure nelle pagg. seguenti):

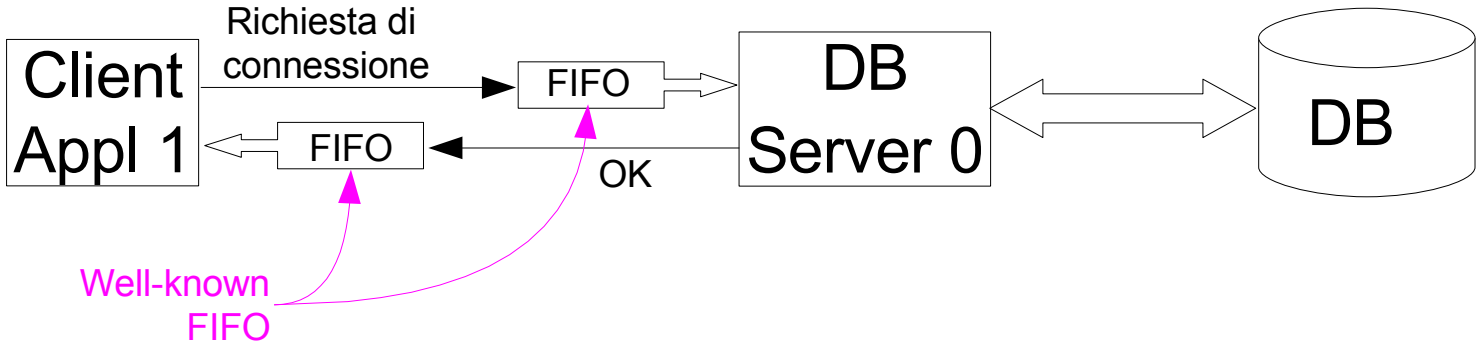
- appena lanciato, inizializzare le necessarie strutture dati ed in particolare creare due well-known FIFO: una di lettura, sulla quale attendere le richieste di connessione da parte dei programmi client, una di scrittura sulla quale ritornare l'esito della richiesta di connessione (OK, ERROR) ed eventuali altre informazioni correlate alla connessione
- se la connessione ha avuto esito positivo, duplicarsi mediante fork() e creare due nuove FIFO dedicate allo specifico client, una per ricevere query e l'altra per restituire i risultati
- servire le query del client fino alla richiesta finale di disconnessione, che comportera' la chiusura ed eliminazione delle due FIFO dedicate al client e la propria terminazione

DB Server (2)

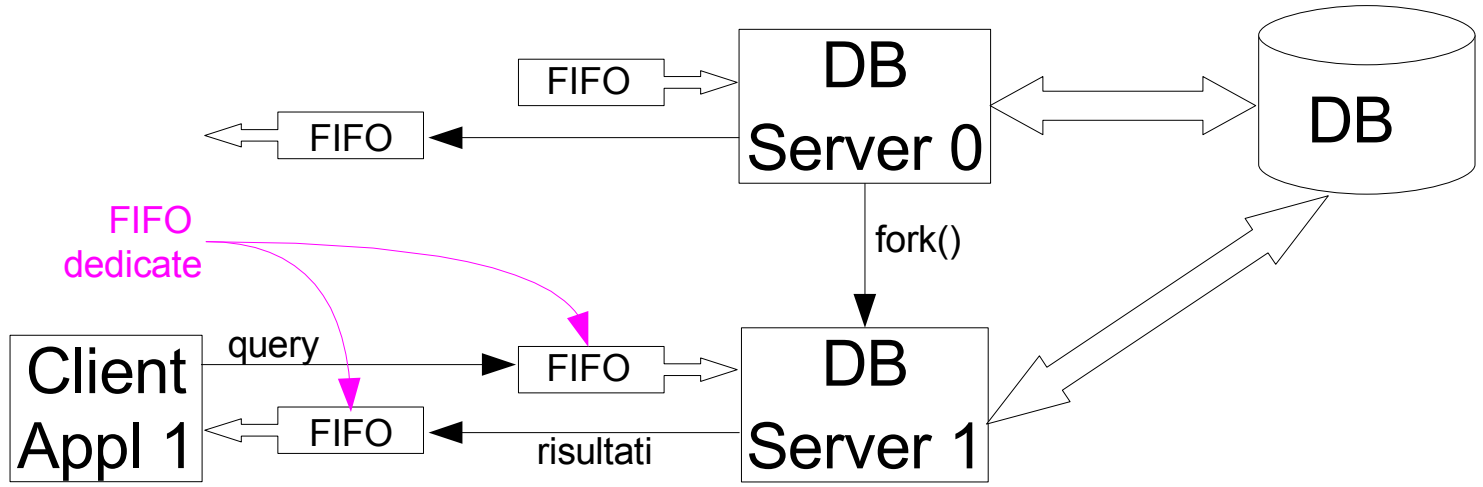
- Ogni nuova istanza di *dbserv* e' identificata da un numerico progressivo
- Il primo *dbserv*, considerato il *dbserv* master (*dbserv 0*), deve mantenere una lista sempre aggiornata dei *dbserv* istanziati e dei nomi delle applicazioni client a loro collegate
- Il singolo *dbserv* lanciato dal *dbserv* master deve memorizzare internamente il proprio identificativo numerico e il nome dell'applicazione client ad esso collegata

DB Server: procedura di connessione

A) Connessione

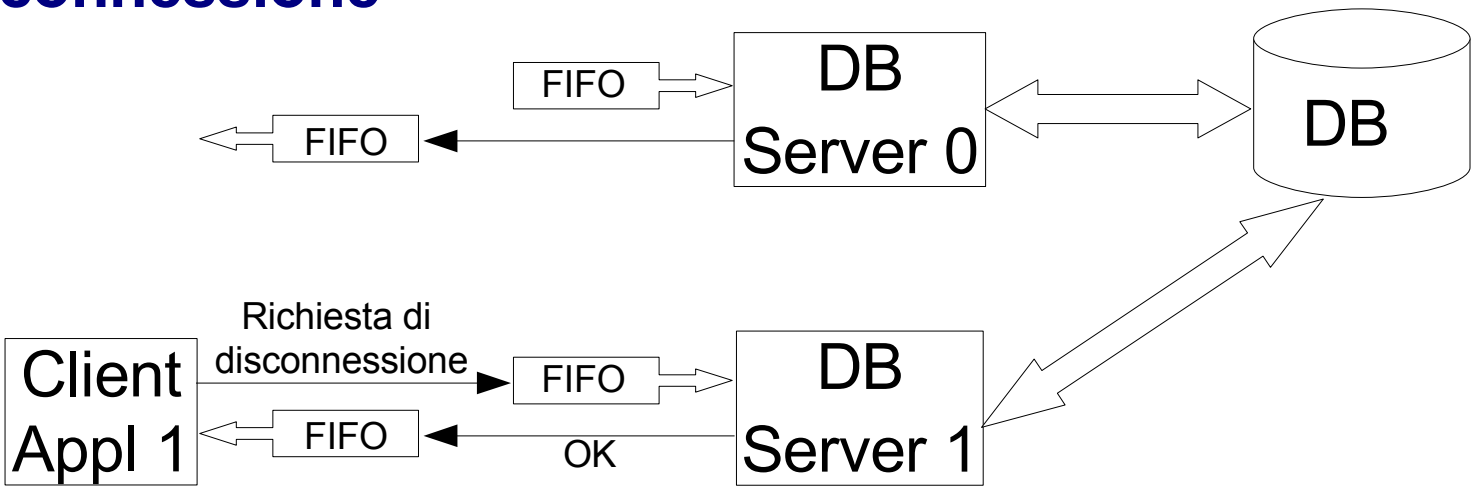


B) Interazione col DB Server

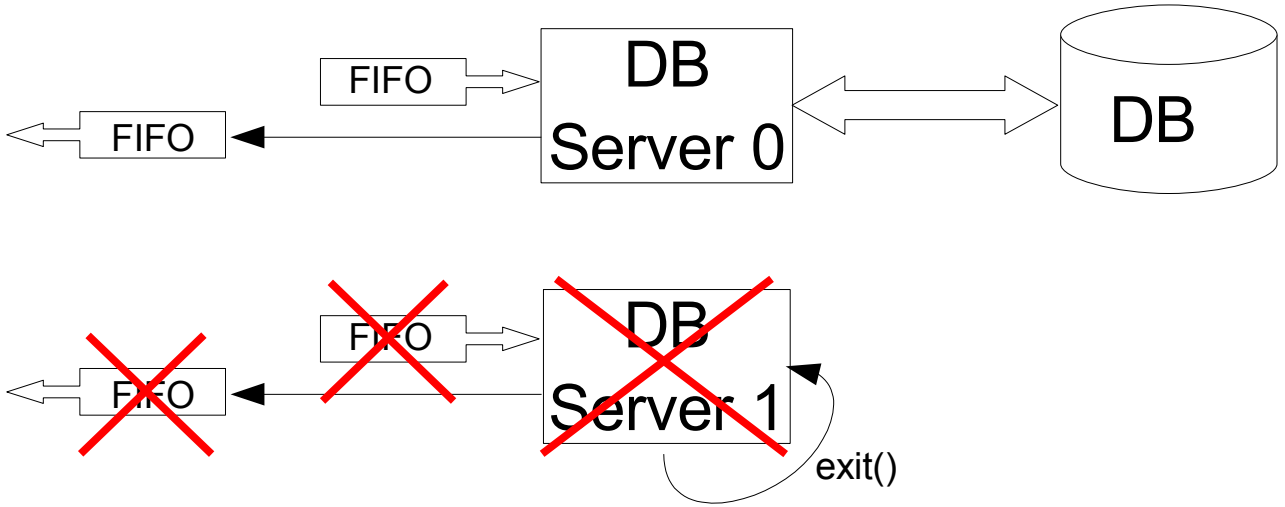


DB Server: procedura di disconnessione

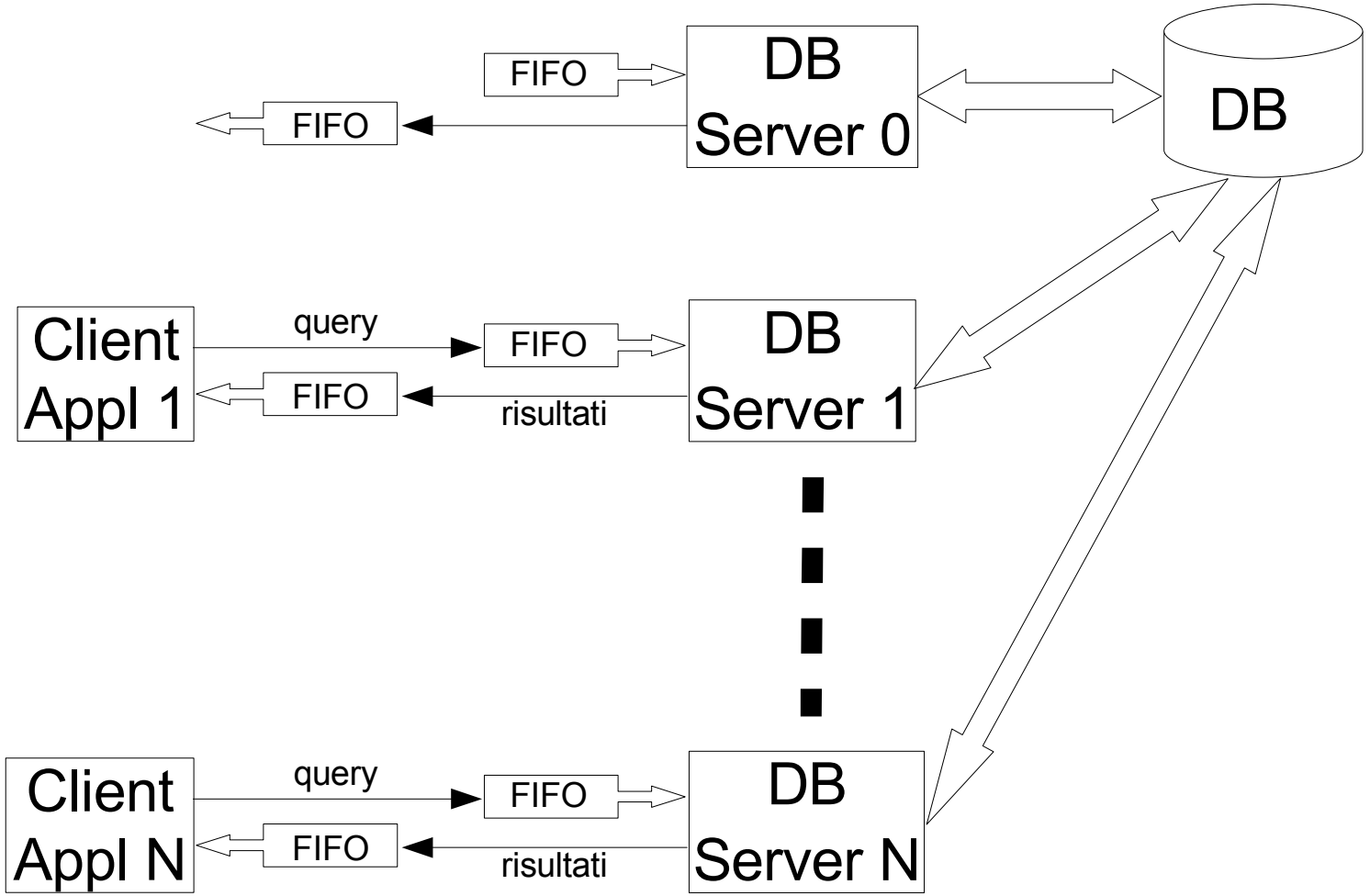
A) Disconnessione



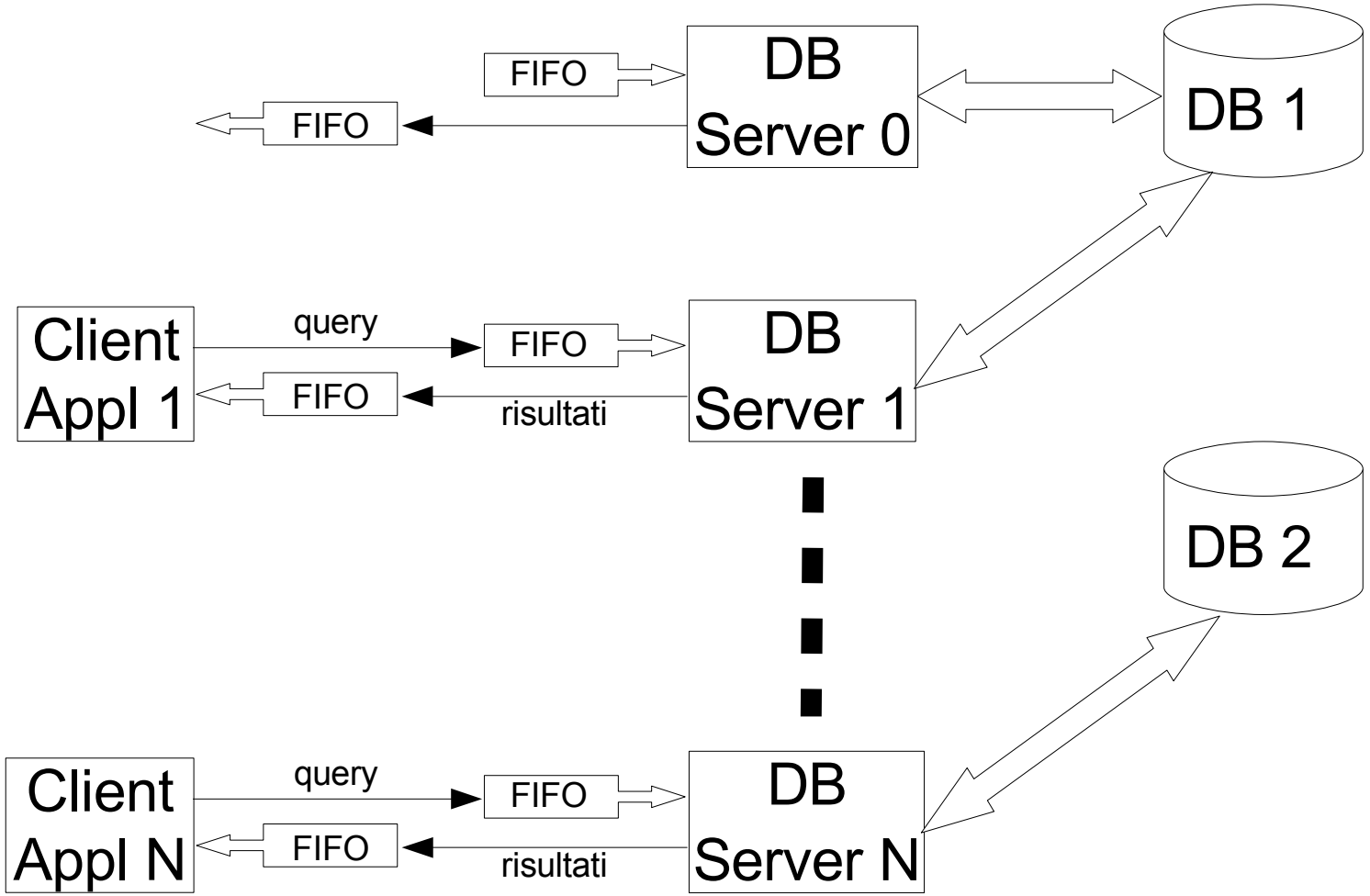
B) Terminazione del DB Server



DB Server: scenario con client multipli e unico DB



DB Server: scenario con client multipli e vari DB



DB Admin

Il programma DB Admin, il cui eseguibile prendera' il nome *dbadm*, deve:

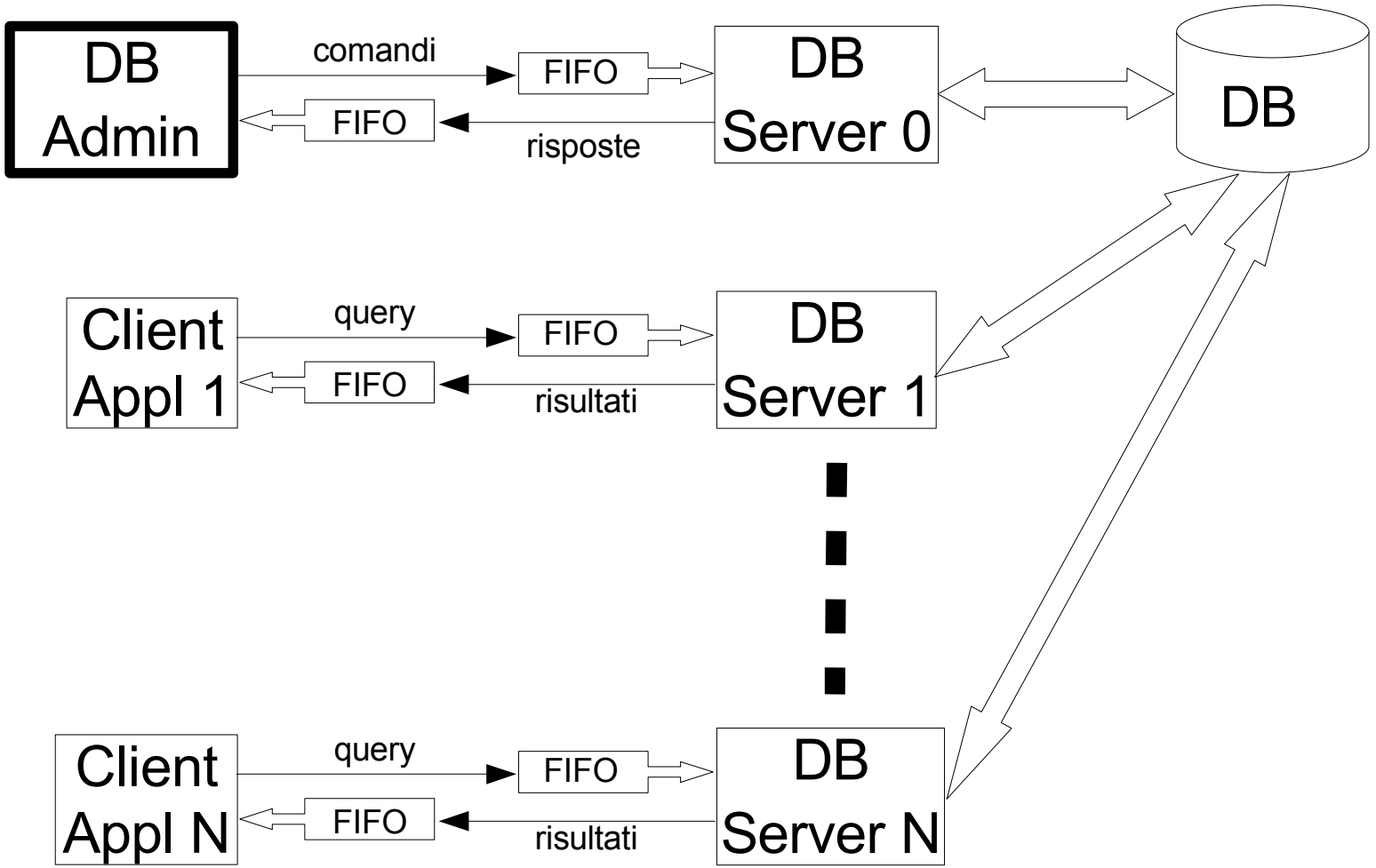
- appena lanciato, verificare se il *dbserv* master e' gia' in esecuzione, altrimenti provvedere a lanciarlo
- comunicare con il *dbserv* master attraverso la coppia di well-known FIFO (quella utilizzata anche per le richieste di connessione dei client)
- restituire un prompt all'utente per servire interattivamente comandi di monitoraggio e controllo del *dbserv* all'interno di una lista predefinita
- terminare la propria esecuzione su richiesta dell'utente secondo 2 modalita': lasciando il dbserv in esecuzione (comando *quit*) oppure terminando il *dbserv* (comando *halt*)

DB Admin: comandi

dbadm deve supportare i comandi:

- **help**: mostra l'help con la lista e descrizione in breve dei comandi
- **quit**: esce senza terminare il *dbserv*
- **halt**: esce terminando tutti i *dbserv* istanziati
- **kill *n***: termina uno specifico *dbserv,* identificato dal suo numerico *n*
- **tables**: mostra la lista delle tabelle del DB (stampa nomi delle tabelle e il numero dei campi in esse contenuti)
- **servers**: stampa il numero di *dbserv* istanziati e il nome delle applicazioni client a loro collegate
- **comandi SQL-like**: si veda la descrizione di DB Lib

DB Admin: interazione col server



DB Lib

La libreria DB Lib, che dovra' essere generata in formato statico col nome *dblib.a,* deve includere le seguenti funzionalita':

- connessione/disconnessione al/dal *dbserv*
- creazione e distruzione di tabelle del DB
- inserimento di nuovi record in una tabella
- modifica di record in una tabella
- cancellazione di record da una tabella
- selezione di campi appartenenti a record all'interno di una tabella

DB Lib: funzioni *connect/disconnect*

- Permettono al client di collegarsi/disconnettersi al/dal DB Server
- La connect deve essere chiamata prima di qualsiasi altra funzione di interazione con il server
- La *connect* accetta come parametro il nome del database (tra quelli gestiti dal DB Server) a cui si vuole accedere:

```
int rc=connect(char *dbname);
```

- La *disconnect* non necessita di parametri:

```
int rc=disconnect();
```

- Statement (pseudo-)SQL equivalenti:

```
CONNECT TO <dbname>

DISCONNECT
```

DB Lib: funzione *create table*

- Crea nel DB una nuova tabella con il nome ed il numero di campi indicato
- Si devono fornire nome e numero dei campi

```
int rc=create(char *tablename, int numfields);
```

- Statement (pseudo-)SQL equivalente:

```
CREATE TABLE <tablename> <numfields>
```

DB Lib: funzione *drop table*

- Cancella dal DB la tabella con il nome indicato
- Si deve fornire solo il nome della tabella

```
int rc=drop(char *tablename);
```

- Statement SQL equivalente:

```
DROP TABLE <tablename>
```

DB Lib: funzione *select*

- Permette l'interrogazione (query) del DB tramite il server, richiedendo l'estrazione di valori da campi selezionati (mediante il loro indice numerico) di una tabella
- Restituisce un vettore di puntatori a stringhe che contengono i valori estratti dai campi selezionati
- L'estrazione dei valori dei campi avviene mediante lettura di quei record che soddisfano la condizione di ricerca (uguaglianza di un campo *chiave* con un valore dato) nella tabella indicata

```
char *values[]=select(int fields[], char
  *tablename, int keyfield, char *keyvalue);
```

- Statement (pseudo-)SQL equivalente:

```
SELECT <fields> FROM <tablename> WHERE
  <keyfield>=<keyvalue>
```

DB Lib: funzione *insert*

- Permette l'inserimento di un nuovo record nella tabella indicata
- Richiede la specifica dei valori di tutti i campi del nuovo record

```
int rc=insert(char *tablename, char *values[], int
   numval);
```

- Statement SQL equivalente:

```
INSERT INTO <tablename> VALUES <v1>,<v2>,...,<vN>
```

DB Lib: funzione *update*

- Aggiorna un campo di un record nella tabella indicata, laddove il valore di un campo assume un valore dato
- Si devono fornire il valore del campo da modificare e la sua posizione oltre alla posizione ed al valore del campo da cercare

```
int rc=update(char *tablename, int fieldno, char
  *value, int keyno, char *keyval);
```

- Statement (pseudo-)SQL equivalente:

```
UPDATE <tablename> SET <fieldno>=<value> WHERE
  <keyno>=<keyval>
```

DB Lib: funzione *delete*

- Cancella un record dalla tabella indicata dove il valore di un campo assume un valore dato
- Si devono fornire posizione e valore del campo da cercare

```
int rc=delete(char *tablename, int fieldno, char
  *value);
```

- Statement (pseudo-)SQL equivalente:

```
DELETE FROM <tablename> WHERE <fieldno>=<value>
```

Hotel (1)

- Il programma deve poter (re)inizializzare il database e crearne uno nuovo
- Il database deve contenere almeno due tabelle:
  - Camere: elenco delle camere con indicazione dell'utilizzo corrente e di chi le sta occupando
  - Clienti: lista clienti (inclusi quelli non più ospiti) con informazioni anagrafiche
- N.B.: e' necessario assegnare un codice che identifica univocamente ogni cliente e che va inserito anche nella tabella delle camere utilizzate (*foreign key*)

Hotel (2)

- L'applicazione deve permettere la gestione completa dell'albergo, almeno con le seguenti funzioni:
  - Registrazione di un nuovo cliente
  - Ricerca di una camera libera per un cliente in arrivo
  - Assegnazione della camera ad un cliente già registrato
  - Rilascio della camera alla partenza del cliente (che però non va cancellato dalla tabella dei clienti)
  - Stampa dell'utilizzo corrente di tutte le camere
  - Stampa della lista clienti

Hotel (3)

- L'applicazione deve supportare l'accesso contemporaneo da parte di più client allo stesso DB attraverso l'istanza del *dbserv* collegata
- Il sistema deve garantire l'integrità del database rispetto agli accessi concorrenti, in particolare rispetto alla registrazione simultanea di due clienti in arrivo, grazie ai meccanismi che *dbserv* e *dblib.a* mettono a disposizione
- Nella documentazione e' necessario indicare in quali casi il sistema (*applicazione*+*dblib.a*+*dbserv*) non garantisce protezione sufficiente
- Ricercare, ove possibile, soluzioni di tipo applicativo (cioe' mediante il suggerimento di sequenze di chiamate alle funzioni della *dblib.a*) alle eventuali carenze del server

Note sull'implementazione (1)

- Il *dbserv* deve poter gestire diversi DB
- Ogni client puo' accedere ad un DB alla volta (non puo' essere connesso a piu' di un DB contemporaneamente)
- Ogni *dbserv* deve utilizzare un meccanismo di file/record locking che garantisca la protezione totale dei dati durante l'accesso allo stesso DB da parte delle varie istanze del *dbserv,* che servono le richieste delle applicazioni client:
  - Lock sulle tabelle durante le operazioni di CREATE e DROP
  - Lock sui record durante le operazioni di lettura e modifica (SELECT/UPDATE/INSERT/DELETE)
- Si devono gestire almeno i seguenti errori (tramite codice di ritorno ove possibile):
  - Errore sui parametri (e.g., parametri mancanti o con valori errati)
  - Tabella non esistente
  - Campo inesistente (e.g., maggiore del numero dei campi della tabella)
  - Impossibile acquisire il lock

Note sull'implementazione (2)

- Il *dbserv*, il *dbadmin* e le applicazioni client girano tutte sullo stesso PC
- Le comunicazioni tra processi devono essere effettuate mediante FIFO (named-pipe) con la limitazione delle dimensioni dei messaggi a PIPE_BUF (4K)
- La well-know FIFO deve essere utilizzata da *dbadm* per l'inoltro di tutti i comandi verso il *dbserv* e dai client soltanto per la prima richiesta di connessione al *dbserv*

Package da consegnare (1)

- Tutte le compilazioni devono essere gestite mediante Makefile
- Il progetto deve essere documentato mediante una breve relazione che descrive le scelte implementative, il dettaglio dei comandi del *dbadm*, le funzioni della libreria *dblib.a* e altre informazioni necessarie all'utilizzo e test del package
- La consegna consiste in un package, da spedire via mail, in formato tar compresso col nome *dbms.tgz* contenente i sorgenti C, gli include file, il Makefile, alcuni script di prova delle funzionalità della console (utilizzando la redirezione dell'input) e la documentazione

Package da consegnare (2)

- L'applicazione client deve essere documentata indipendemente dal DBMS (*dbserv*+*dbadm*+*dblib.a*) e deve essere installabile e compilabile indipendentemente dal DBMS
- Il DBMS deve avere include file e librerie distinte da quelli dell'applicazione e deve essere installabile indipendentemente
- Devono essere presenti script di test comprendenti comandi SQL-like da sottoporre (tramite redirezione dell'input: **`dbadm < scriptfile`**) alla console per la verifica di tutte le funzionalità
- Il compito di esonero deve essere compilabile ed eseguibile su distribuzione RedHat/Fedora Core o Suse, utilizzando GCC v3

Estensioni

- Il compito consegnato può contenere estensioni alle specifiche minime richieste, per esempio:
  - Supporto ad altri tipi di variabili (stringhe di lunghezza variabile, interi, etc.)
  - Supporto ai nomi di colonne (oltre che al numero)
  - Possibilità di avere più server attivi simultaneamente sulla stessa macchina
- Tutte le estensioni debbono essere documentate ed utilizzate dall'applicazione fornita