FSDB

Documentation technique

Sébastien Rastegar – Guillaume Estocq – Ludovic Laguerre –
Michel Riedel – Loïc Doyen

EPITECH INNOVATIVE PROJECTS

www.fsdb.fr
www.epitech.eu

A propos de ce guide :

Le but de cette documentation est de permettre à des développeurs de comprendre le fonctionnement de FSDB pour continuer à travailler sur ce projet.

Pour cela, elle expliquera sous forme de diagrammes les différentes interactions entre le système et FSDB. Ce document est complété par le guide utilisateur qui s'adresse aux utilisateurs finaux.

Historique des versions :

Version	Date	Modification
1.0	04/04/2008	Création
2.0	21/10/2008	Refonte graphique et mise à jour du contenu.
3.0	05/01/2009	Ajout des limitations et des bugs, du référentiel et refonte complète des autres chapitres.

Table des matières

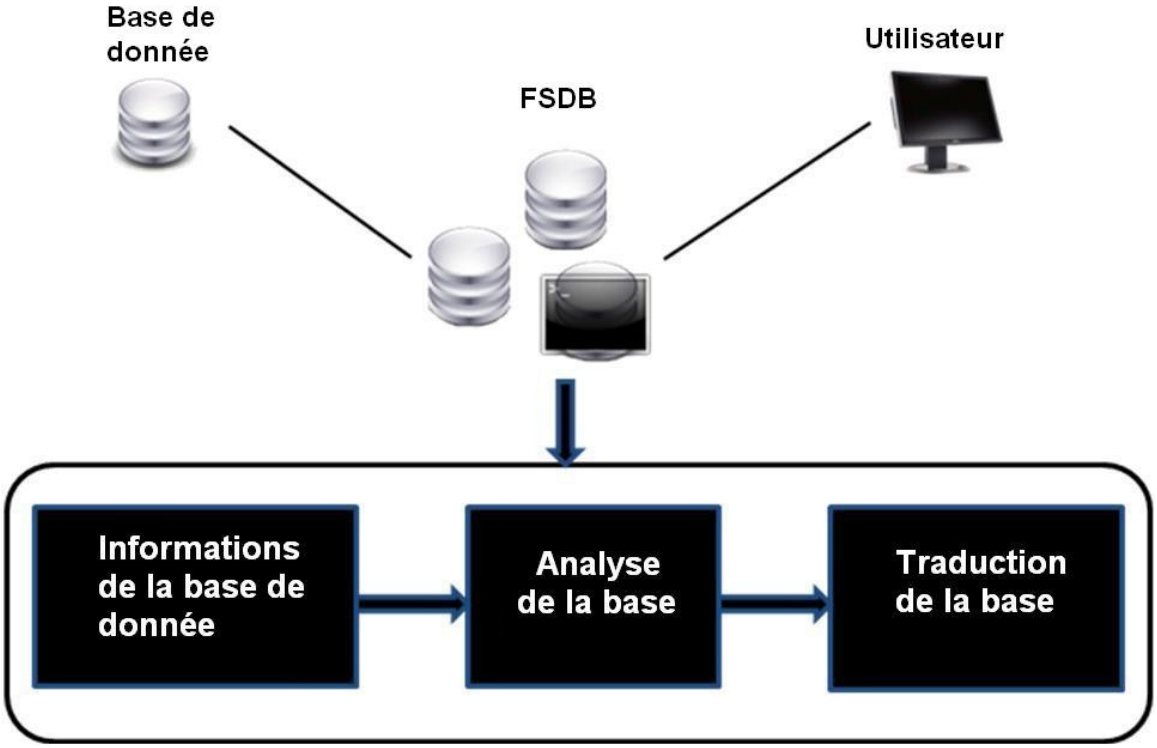
I – Présentation générale :

FSDB* est un traducteur de commandes. Ce projet a pour but principal de permettre à une personne de gérer une base de données dont elle aura au préalable imaginée ou recopiée la structure, grâce aux commandes shell qu'elle a appris pour gérer son système de fichiers ou avec l'explorateur de fichier de son système d'exploitation basé sur Linux.

Le but principal réside dans l'aspect conceptuel, à savoir que la finalité du projet est de faciliter l'utilisation d'une base de données, tout en offrant un niveau de performances au pire identique à l'existant.

La base de données pourra ainsi être montée, à la manière d'un système de fichiers, et être gérée comme tel par l'utilisateur, avec tous les avantages que cela implique, l'absence de phase d'apprentissage d'une nouvelle syntaxe, et l'appel à l'interpréteur de commandes dans le cadre de la programmation, fonction implémentée dans la quasi totalité des langages.

II – Fonctionnement général :



Ce diagramme montre les différentes étapes nécessaire à FSDB pour traiter une base de données.

Informations de la base de données	Cette étape correspond au passage des paramètres pour que l'application s'exécute correctement.
Analyse de la base	Cette étape correspond à l'information donnée par l'utilisateur concernant le type de base de données.
Traduction de la base	Cette étape concerne l'exécution du programme.

III – Prérequis :

Notre application est utilisable dans un environnement Unix/Linux mais est aussi compatible avec n'importe quel type de base de données (MySQL, Postgre, Oracle), tant qu'une librairie est implémentée afin d'assurer l'inter-polarité entre FSDB et le type de base utilisée.

IV – Technologies utilisées :

1. Langages utilisés :

- C/C++ : Dans un premier temps le projet a été développé en C, puis dans un souci de modularité, il a été décidé de migrer le code en C++
- SQL : Langage permettant de manipuler une base de données.
- Perl / Gtk 2 : Ensemble de librairies permettant de réaliser une interface graphique.

2. Outils utilisés :

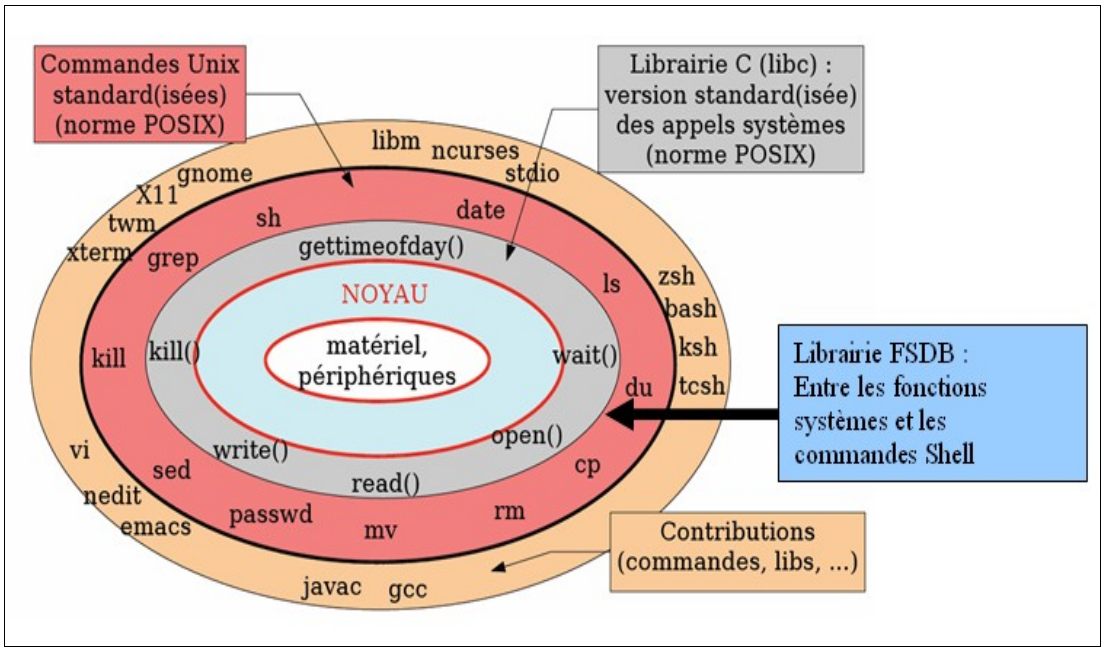
Afin de réaliser notre projet, nous avons eu besoin d'outils pour sa conception, ses tests, et sa mise à disposition du public.

- Eclipse : Il s'agit d'un environnement de développement qui nous a permis de coder notre projet.
- Automake : C'est un générateur pour compiler notre programme, il nous a permis un gain de temps lors des phases de test quand nous voulions corriger d'éventuels bug.
- Esvn : C'est un client svn qui nous a permis de partager les sources et les documentations de FSDB.

V – FSDB dans son environnement de travail

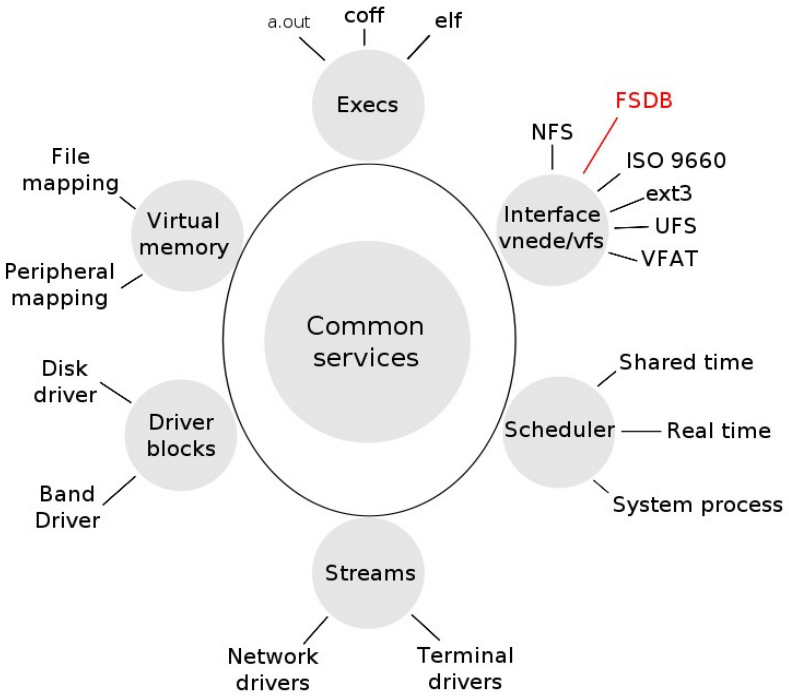
1. FSDB au sein du système d'exploitation

FSDB est un programme qui interagit directement avec le système, c'est pourquoi il est important de situer et connaître l'utilité des différentes couches de son système d'exploitation Unix pour comprendre le fonctionnement de FSDB.



Comme dit dans la présentation générale, FSDB est un traducteur de commandes, c'est pourquoi il n'est pas étonnant de trouver la librairie FSDB entre la couche des commandes UNIX et la couche comportant la librairie C. Ainsi il est possible d'associer une commande shell aux fonctions de FSDB qui utiliseront la librairie C.

2. Interaction avec les services du système d'exploitation



Un système d'exploitation fournit un certain nombre de services qui permettent l'interaction entre celui-ci et les programmes. Ainsi nous voyons que FSDB communique avec l'interface vfs (Virtual File System).

3. FSDB et les librairies

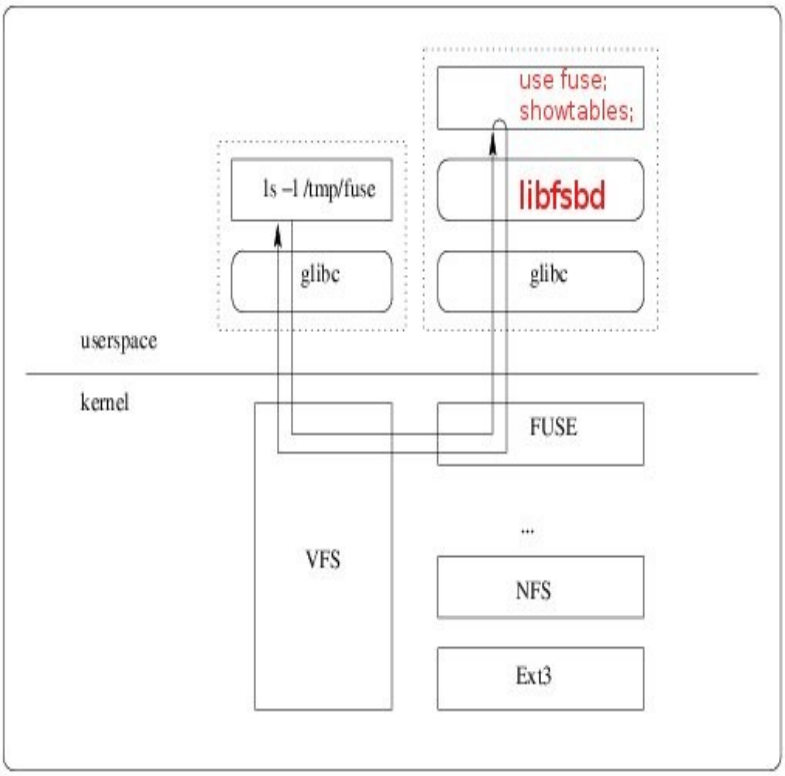
Nous avons vu que FSDB se situe en marge de la librairie C. En effet, pour monter une base de données, FSDB utilise des appels dans les langages fournis par les API des différentes base de données. Cependant, pour monter une base de données, FSDB utilise FUSE.

FUSE permet de créer un point de montage ou nous permet de mettre en place notre filesystem. FUSE offre une API qui propose des outils pour la définition des appels système.

Nous allons donc définir les principaux appels système dans un format pré-défini donné par l'API FUSE. Nous pouvons changer les propriétés de ces commandes dans la requête SQL à l'aide de la libsql.

Toutes les commandes inutiles sont désactivées.

Ce diagramme montre les flux de communication entre FUSE et les différentes librairies qu'utilise FSDB.



Ainsi, ce diagramme des flux montre les librairies parcourues lorsque l'utilisateur fait un LS ce qui correspond à un show tables en SQL.

VI – Limitations et bugs connus

Une série de tests on été effectués pour tester la stabilité de FSDB, ses limites et trouver d'éventuels bugs.

Les tests suivant ont été réalisé avec des commandes couramment utilisées dans la gestion de bases de données :

Test 1: Affichage d'une table

Le test effectué consiste à répéter plusieurs fois la commande. Cela permet de voir si FSDB est robuste à la répétition de commandes et suffisamment rapide à leurs exécutions.
Durant ce test, le même script a été lancé à partir d'un autre terminal, ce qui permet de vérifier qu'il n'y ait pas de problème d'accès concurrentiel.

Résultat : Aucun problème relevé

Test 2: Création d'une table

L'objectif de ce test est de montrer la stabilité et la fiabilité du programme en faisant un script qui va créer à la suite 400 tables de 2 colonnes dans la base.

Résultat : OK, à noter que le traitement peut être long,suivant la puissance du serveur ou la méthode utilisée pour héberger le serveur (sur une machine virtuelle par exemple).

Test 3: Création/suppression d'une table

Nous testons l'exécution de 2 scripts identiques lancés en parallèle, qui cherchent à supprimer la même table et à la recréer. Ce test permet de voir si FSDB gère bien les accès concurrentiels.

Résultat : OK, il est impossible de créer ou supprimer la même table lorsqu'un utilisateur est en train de travailler dessus.

Test 4: Connexion à la base

Nous allons essayer de nous connecter à la base avec un nom d'utilisateur qui n'existe pas sur le serveur.

Résultat : OK, le shell nous renvoie une erreur 'Permission non accordée', même en mettant tous les droits sur le dossier virtuel on ne peut y accéder, seul le root et les utilisateurs de la base peuvent consulter ce dossier.

VII – Le multibase de données

Notre projet gère lors de l'installation MySQL et PostGre, cependant un système de librairies partagées est mis en place afin de permettre la compatibilité au plus grand nombre de types de bases de données tant qu'elles possèdent une librairie C.

Ce système permet de rendre notre projet évolutif, ainsi chacun peut implémenter sa librairie pour utiliser FSDB dans la gestion de son type de base préféré.

En annexe 1 est fournie l'API pour interfacer sa librairie avec FSDB.

VIII – Référentiel

Document	Acronyme	Status
Cahier des charges	CDC	OK
Étude de l'existant	EDE	OK
Rapport d'étude détaillé	RED	OK
Document d'architecture	AA	OK
Documentation technique	TD	OK
Guide utilisateur	UG	OK
Compte rendu d'activité	CRA	OK
Plan d'assurance qualité	QIP	OK

Une partie de ces documentations sont disponibles sur notre site internet : www.fsdb.fr . Pour avoir accès aux autres documentations ou toute autre suggestion, merci de nous contacter à partir du site internet.

ANNEXE 1 : API pour implémenter une librairie partagée pour un type de base de données

```
/*
 * CDBlib.h
 *
 *  Created on: 7 oct. 2008
 *      Author: ludo
 */

#ifndef IDBLIB_H_
#define IDBLIB_H_

#include <map>
#include <string>
#include <vector>

#include <syslog.h>

#include <SQLQuery/SQLQuery.h>

typedef struct s_off{
 unsigned long off_row;
 unsigned long off_str;
}t_off;

typedef std::map<std::string, std::string > ConnDB;
typedef std::map<std::string, std::string> colDesc;
typedef std::map<std::string, colDesc > tblDesc;
typedef std::vector<std::string> tabString;

using namespace std;

class IDBlib {
public:
 virtual void connectDB(const ConnDB&) = 0;
 virtual void closeDB() = 0;

 virtual void getTblDesc(const string&, tblDesc&) = 0;
 virtual void getColDesc(const string&, const string&, colDesc&) = 0;
```

```
 virtual bool dropColumn(const string&, const string&) = 0;
 virtual bool dropTable(const string&) = 0;
 virtual bool addColumn(const string&, const string&, const string&) = 0;
 virtual bool createTable(const string&) = 0;
 virtual bool renameColumn(const string&, const string&, const string&) = 0;
 virtual bool renameTable(const string&, const string&) = 0;

 virtual void showTables(tabString&) = 0;
 virtual void showColumns(const string&, tabString&) = 0;

 virtual unsigned long fileSize(const string&, const string&) = 0;
 virtual unsigned long fileSize(const SQLQuery &query) = 0;

 virtual bool tableExist(const string&) = 0;
 virtual bool columnExist(const string&, const string&) = 0;

 virtual int  readColumn(const string&, const string&, char *, size_t, off_t, t_off*) =
0;
 virtual int  readColumn(const SQLQuery &query, char *, size_t, off_t, t_off*) = 0;
};

typedef IDBlib* (*create_t)(const ConnDB&);
typedef void (*destroy_t)(IDBlib*);

typedef struct s_dblib {
 void  *handle;
 create_t create;
 destroy_t destroy;
}fsdb_struct_dblib;

#endif /* IDBLIB_H_ */
```