A CONFIGURABLE LINUX FILE SYSTEM FOR MULTIMEDIA DATA

Nicola Corriero, Vittoria Cozza, Eustrat Zhupa
Dipartimento di Informatica, Università di Bari, Via Orabona 4 - 70125, Bari, Italy

Vito De Tullio
Dipartimento di Informatica, Università di Bari, Via Orabona 4 - 70125, Bari, Italy

Keywords: Fuse, file system, tag, configurable, Linux, mesh network.

Abstract: In MusicMeshFS the tree structure of the virtual Linux filesystem, extended and made configurable by the MusicMeshFS language, is adapted for storing and efficiently retrieving multimedia data. In the case of installing MusicMeshFS inside an embedded system equipped with WIFI card, the multimedia data sharing over an ad hoc mesh network can be achieved for free.

1 INTRODUCTION

MusicMeshFS (MMFS) is here introduced as a system able to classify and to manage multimedia files collections of growing size, in such a fast automatic way, configurable and portable on different platforms. MMFS appears as a virtual file system, implemented on Linux in user space, where multimedia files are the data source and a directory tree, arranged according to end user preferences, represents the output. As for every Linux file system it's possible to perform on it classical operations as read, open, rename and so on and all these operations will be based mainly on music tags (for example author, album) values. It's either possible to use the shell tools like *find* to query the meta information of multimedia files. This approach enables the user to keep handy collections of data as well as meta information in order to reorganize data according his own choices. With meta information we mean the tag one can associate to each file in almost every audio format.The tags we consider in MMFS are information such *author of the track*, *title*, *year*, *album*, *track number*, *musical genre*, that can be extracted by almost every musical format thanks to the *TagLib* C/C++ library (Wheeler, 2008). MMFS has been implemented in C language and distributed under the GNU GPL license [1]. The project is based on the use of the FUSE library (Szeredi, 2008, Singh, 2006) to generate a file system in user space.

[1]http://code.google.com/p/musicmeshfs/

2 A MUSIC FILE SYSTEM IN USERSPACE

The main idea of this work is the managing of music information by a file system tree, using the FUSE library. It avoids the necessity of working in kernel space and we get rid of the portability problems between the various releases of the Linux kernel. Instead, we can use a simple API to implement a file system with a series of callbacks depending on the operations to perform. FUSE appears to the kernel like a classical file system, but its behaviour is defined in user space. Implementing such file system requires some extra cost in terms of computation. Anyway, it's a fair price to pay for the advantages it offers: all meta information of the multimedia files are available for access by shell tools or even by graphical interfaces (file browsers).
The use of FUSE library in the project is limited to the implementation of some of the callback functions defined in the `struct fuse_operations`. These callback functions are self explaining for a Linux user, as getattr, read, readdir, readlink, mknod, mkdir, unlink, rmdir, symlink, rename, link, chmod, chown, truncate and write.
Furthermore, there are some callback functions im-

Corriero N., Cozza V., Zhupa E. and De Tullio V. (2008).
A CONFIGURABLE LINUX FILE SYSTEM FOR MULTIMEDIA DATA.
In *Proceedings of the International Conference on Signal Processing and Multimedia Applications*, pages 380-383
DOI: 10.5220/0001937103800383
Copyright © SciTePress

plemented to optimize the code execution.
For semplicity, has been developed a little library over FUSE, called `db_fuse`, where a subset of these functions has been implemented, with the purpose to offer all the necessary functionality to manage tags of musical files. `db_fuse_getattr()`, `db_fuse_readdir()` and `db_fuse_read()` have been the first functions implemented, because they enable navigating inside the file system. Particularly `db_fuse_readdir()`, must work with a user schema considering it a generative grammar to populate the directory(Love, 2005). Then `db_fuse_open()` and `db_fuse_release()` have been added to the project; anyway, because generally multimedia files have a big size, and they are read in chunks by players, it doesn't make sense that at each access the player must resolve the real file name; the idea is that only at the moment of opening the virtual file for the first time, `db_fuse_open()` analyzes the path name of the file, finds the first real file matching the user requirements, opens it in readonly mode and it refers to this file using the returned file handler. `db_fuse_read()`, then, will just use the file handler, and `db_fuse_release()` will refer to it for closing the file. In conclusion, `db_fuse_rename()` has been realized with the aim of invoking `rename_local_file()`, once done the syntactic controls over virtual path about the name to modify and the new expected name.
Actually, the role of `rename_local_file()` is to set tags with new values inside the multimedia file. Implicitly the database will be updated too, this because the monitoring process will assure real time synchronization of the database with new data when new tags are inserted. This is possible thanks to *inotify*, a Linux kernel API that makes simpler the monitoring of the changes in files and directories. Inotify supports event-driven programming; it makes possible that when directories and files are added, updated or removed from the pool of monitored directories, the new files will be parsed and the database will be updated.

3 MMFSLANGUAGE

MMFS makes possible the indexing of musical collections and showing of data stored inside the database by a directories view, as many existing system do, but moreover it offers to the end user a configuration language, that allows to query the database without the need of SQL-like languages knowledge. From the user point of view, MMFS can be used as a classical file system, by command line and by graphical file browser as well. In addition it allows to choose the multimedia files organization by setting a client side parameter that is called *schema*.
By the schema the user can set the final structure of directories by using some placeholders, for example %artist or %album, to tell the system that the file name or directory name will contain, in that position, music belonging to the artist or to the album.

The language to set the schema will follow the Extended Backus-Naur Form grammar:

```
<SCHEMA> ::= <NAME> ( "/" <NAME> )*
<NAME> ::= <NAME1> | <NAME2>
<NAME1> ::= <FIXED> ( <NAME2> )?
<NAME2> ::= <KEYWORD ( <NAME1> )?
<FIXED> ::= ( [^%/] | "%%" )+
<KEYWORD> ::= "%artist" | "%year"
| "%album"| "%track" | "%title"
| "%genre" | "%host"| "%path"
| "%type"  | "%filename"
```

Generally talking, a schema is divided into directory and file name, each of them represented as a sequence of keywords and fixed elements. Keywords and their semantic must be defined in a configuration file that will be used inside `db_fuse` library.
The schema aims to give the possibility of introducing an order relation amoung source files: a conceptual order with respect to file system tree levels, a lexicographical order among sibling nodes. In figure 1.*a*

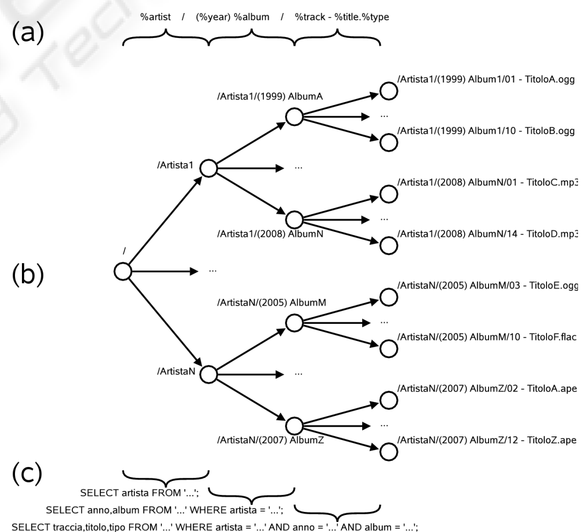

Figure 1: Schema.

you can see an example of schema inserted by the final user and in 1.*b* the resulting tree. The root obviously is '/' (this example is suppose to follow the file system mount point). The schema can be divided into levels, each separated by the directory separator character ('/'). Then `%artist(%year) album/%track`

`- %title.%type` must be considered as composed of three parts: `%artist`, then `(%year) album` and finally `%track - %title.%type`. In this simple example the whole first level is just a keyword: `%artist`, then implicitly the user is asking for all the artist names belonging to the database. The system then will perform a query such as "SELECT artist_name FROM ...;" where "artist_name" is the table corresponding to the keyword "%artist", figure 1.*c*. The sublevels will be calculated in a similar way, but using the WHERE clause too, imposing stronger restriction going deeper in the tree. Each couple "database column", "value" corresponds to the variable terms found inside the hierarchy at that time. Then, to have the enumeration of files presents in the directory - let's say "U2" - will be performed a query such as "SELECT year, album FROM ... WHERE (artista = "U2");" and so on.

4 MUSICMESHFS ARCHITECTURE

The modular architecture of the system can be seen mainly as formed by a dæmon and a client. Musicmeshfsd is the process that aims to ensure consistency among real data (information inside monitored files and directory) and the internal database. More in detail, it has to monitor local files but it can be extended to intercommunicate with other instances of musicmeshfsd in other hosts so to work over remote files.
Musicmeshfsc is the process that makes possible to show informations like in a file system tree according to a user defined schema. Then the tree will be populated with the information stored by musicmeshfsd.
The file system is implemented in quasi-read-only mode, as it won't be possible to create or to move external multimedia files, and the only reasonable modification consists of renaming. This because semantically it doesn't make sense deleting or creating a virtual file. The reason why renaming is allowed is that it can be useful to change the file, in the sense of renaming its tag. Obviously, it's not possible to move a file inside a virtual file system because there is not any physical place (source directory) where to move the file. Remind that the file collection can be established just adding files to the monitored directory and then modifying the database. A nonsense case is the one of creating an empty file. In this case the user will create just a name in the database not associated to any real file. The role of renaming is important for enabling the end user to correct tagging mistakes or eventually to add missing information. So rename acts only on the tag and is implemented in such a tricky way that is possible to move directories and files without limitation, but the new file name must contain the same information with the original file. Obviously, the destination file will follow the user schema and, even more, the depth level between old and new name must be the same to avoid ambiguity between the tag types or having an inconsistent database. In figure 2 the general architecture is showed. The two main components intercommunicate by a shared relational database. Musicmeshfsc implements the previously analyzed FUSE functions, that correspond to a realization of the Linux Virtual File system Switch.
The usage of a database to manage information, appears as a solution that makes our file system able to manage any kind of information that includes metadata. Currently the db_fuse library manages data once they are stored inside the database. The same approach has been used as well to arrange in a file system fashion metadata retrieved from emails (data, object, subject, sender, receiver and so on).

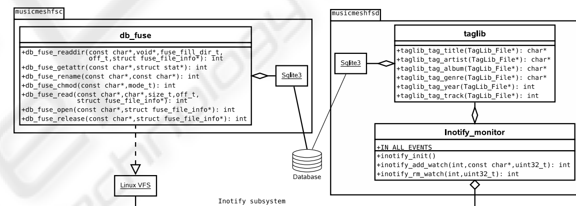

Figure 2: Architecture.

In the project's architecture there is a component included in the musicmeshfsc, that represents a reusable and extendable library that can be used for general purpose application to create a virtual file system for any kind of data stored inside a sqlite3 database.
Musicmeshfsc has been divided in two well separated logical parts: one database content dependent, the other not.
First part accomplishes generation of the file system hierarchy, starting from a generic database and a configuration schema; the second part accomplishes management of insert, update and remove operations refered to single tuples inside database tables, as well as the interpretation of keywords choosed by the end user.
The user must define the database table names, the keywords to use inside the schema to arrange the file system and the related table column name and finally in case of JOIN the tables that is possible to merge and the constraints to impose in the WHERE statements (T1.externKey == T2.referencedValue). Once

the user sets these values, it will be easier to execute the right query for the information retrieval.
Inside the db_fuse library two parsers have been implemented, one for the configuration schema, the second to parse the virtual file system pathnames. These features will allow the user of this library to be able to configure the file names by a schema.

5 MUSIC SHARING OVER EMBEDDED DEVICES

Every embedded device can use Linux and MMFS. One way to boot a kernel on mobile devices by a non-invasive approach is showed in (Corriero et al., 2008); the idea is to install the kernel over a miniSD card, using *Haret*(REF, 2007) as bootloader.
The compilation kernel phase appears quite tricky because of special configurations and optimized patches for the used device. The kernel version used is *2.6.21.hh18*, where *hh* stands for handhelds. The source code was already patched with needed modules for htc blueangel family devices[2]. Compiled the kernel, a minimal embedded distribution has been created. Embedded devices offer technologies well suited to implement wireless versatile functionalities for creating ad hoc mesh network; The idea is to build up a decentralized ad hoc mesh network without static servers. In this kind of network is possible to reach far nodes inside the network exploiting the shared band among all network nodes. The nodes then will act both as clients and as servers. Generally wireless devices are used for managed connection, in the sense that there is an access point to allow a node to connect to network services. In this work an ad hoc connection is proposed but in the opposite it's possible to create a peer to peer communication without the need of a server node and then it's possible to create a mesh network infrastructure; depending on the kind of connection, the way of configuring the wifi card changes. In all the case where mobility plays an important rule (smartphone), it's better to consider a special case of Wifi Mesh network in mobility context linked in ad hoc manner: the manet. Routing protocols refering to manet take into consideration the high level of mobility of network nodes, particularly the Ad hoc On Demand Distance Vector reactive protocol (aodv) generates a path between nodes only on a source node demand. To allow a smartphone to natively use mesh network, the *aodv-uu* module(aod, 2008) has been included while compiling the kernel. Installing MMFS on such embedded devices in the manet, brings to the creation of a framework for sharing musical information among mesh nodes. The virtual music file system will be able to managing, analyzing and monitoring a set of directories and extracting multimedia data from files of supported types and offering a file system view of these data inside a directory tree arranged accordingly to the user preferences. One open problem is that of shared music author copyright. A system to certify licenses is needed to avoid the spreading of illegal detected music by any network node.

[2]To download hh-kernel source code:
```
cvs -d :pserver:anoncvs@anoncvs.handhelds.org:/cvs
checkout linux/kernel26
```

6 CONCLUSIONS AND FUTURE WORKS

Managing the music data inside a virtual file system seems to be a perfect solution for users that must hang with a big musical data bank and that don't accept the static data organizations proposed from almost all the music players. As well, it seems to be a good way for managing the organization of exchanged and shared multimedia files coming from different sources inside a mesh, hiding to the end user the classification but allowing him to impose his own hierarchy. A further extension of this work would be to extend the number of analyzed tags and to insert personal metainformation, for example about personal opinions of the user refering to a music track, generating an always original music tree.

REFERENCES

(2007). Haret reverse engineering tool.

(2008). Aodv-uu homepage - uppsala university. http://www.it.uu.se/research/group/coregroup.

Corriero, Cozza, Pistillo, and Zhupa (2008). Wifi mesh for handhelds in linux.

Love, R. (September 2005). Kernel korner - intro to inotify.

Singh, S. (2006). Develop your own filesystem with fuse. http://www.ibm.com/developerworks/linux/library/lfuse.

Szeredi, M. (2008). Fuse. http://fuse.sourceforge.net. Home page.

Wheeler, S. (2008). Taglib, http://developer.kde.org/wheeler/taglib.html. Home page.