PROJECT REPORT

# MULTIMEDIA FILE SYSTEM

SUBMITTED IN PARTIAL FULFILMENT OF THE DEGREE OF

BACHELOR OF TECHNOLOGY

*by*

**Ajith Prasad B**
**Ibrahim A**
**Sujith P**

*Under the guidance of*

**Saidalavi Kalady**

**2004**
**Department of Computer Engineering**
**National Institute of Technology, Calicut**

**National Institute of Technology, Calicut**
**Department of Computer Engineering**

*Certified that this Project entitled*

# MULTIMEDIA FILE SYSTEM

*is a bonafide work carried out by*

**Ajith Prasad B**
**Ibrahim A**
**Sujith P**

*in partial fulfilment of their*
*Bachelor of Technology Degree*
*under our guidance*

---

**Saidalavi Kalady**
*Faculty*
*Dept. of Computer Engineering*

**Dr. V.K. Govindan**
*Professor and Head*
*Dept. of Computer Engineering*

## Acknowledgement

We thank our guide,Saidalavi kalady for his whole hearted co-operation and for the informative discussions we had in the course of our project.We also thank our classmates,especially Ajin George Joseph for helping us to get past many difficult problems in the compilation phase.We also thank the CSED lab staff for providing computer facilities.

**Ajith Prasad B**
**Ibrahim A**
**Sujith P**

## Abstract

Traditional filesystems are designed primarily considering files having small file sizes.In this age of multimedia systems, it is imperative that a filesystem is built for files having large storage sizes.Through this project,we have bulit such a filesystem.Our filesystem resides in an independent partition and is conformable to the VFS interface in linux.Presently it runs under the linux operating system.Though not provided with all the facilities of a modern filesystem, this can be used as a base for developing more powerful multimedia storage systems.

# Contents

# Chapter 1

# Introduction

As computing world proceeds to the next generation,computing systems tend to become more and more specialised.Specialised gaming computers,weather prediction systems, server systems etc are coming into the market.So there is marked trend towards specialisation; both at the hardware and software level.In this context we have designed and implemented a filesystem targeting multimedia applications.

## 1.1 Overview

This report initially focusses upon the meaning,reason and background of a multimedia filesystem.Then we proceed to the development platform and environmental details.After that we get into the design and implementation part of filesystem.Then it covers the significant improvements and design features that we have introduced aiming at performance enhancement. After that we discuss the limitations and the unresolved bugs,that have crept into our code.Then we conclude by suggesting the possible improvements further works possible on this system.

# Chapter 2

# Meaning Of Multimedia Filesystem

A lot of people could not comprehend the true meaning of a MultiMedia FileSystem ( MMFS ) when we introduced this idea initially.We would make it clear here itself.

## 2.1 Filesystem - Operating System Dependence

Filesystem is commonly viewed as an inseparable part of operating system.But this perception is not always true.Filesystems like FAT and ext2fs are supported under multiple platforms,though they are optimised for windows and linux respectively.

Essentially,MMFS is an attempt to implement an OS independent filesystem, catering to the storage requirements of large volume files like jpeg,mpeg,divx etc.But since we require a running operating system for coding,we have worked upon linux,conforming to its Virtual File System ( VFS ) interface, which will be explained later.

## 2.2 Where MMFS Runs ?

An independent hard disk partition can be formatted with this filesystem.It is coded according to the specifications and constraints of VFS.It can be mounted on some directory and used like any other filesystem.

Like the explore2fs program in windows,once someone writes an explorer for MMFS from other operating systems,this filesystem can be used from those operating systems as well.MMFS provides a generic interface comparable to other existing filesystems. It can run in unison with other filesystems like fat,ext3fs,ext2fs under linux.

# Chapter 3

# Why A Multimedia Filesystem ?

There are a number of reasons behind our decision to work on a project like this.It pertains to historical,technical and industrial factors.We will look into those here.

## 3.1 Limitations of Traditional Filesystems

Initially filesystems were developed as part of operating system.In the early days of computing,storage space was scarce and the files at that time barely exceeded a few kilobytes in size. The current explosion in size of files started in later part of 1990s.At that time,the filesystems were optimised for small file sizes.Eventhough improvements were made upon filesystems after the advent of multimedia technology,the legacy of small sized files were firmly thrust upon those filesystems,particularly ext2fs.

To have an idea upon the inefficiency of ext2fs on multimedia files,let us look into their operation.To access a file from ext2fs, we have to fetch the inode first,calculate the block address from the inode entries and then get the contents from the disk block. So every block access actually consists of two disk accesses. Equating this with the facts that the block size of ext2fs is 1KB and the average size of a JPEG file is about 36KB,we can infer how inefficient is it.

## 3.2 Industrial Demands

As the entertainment and gaming industry develops,computing systems dedicated to such purposes will unfold;filesystems being no exceptions.Also, a crossplatform tool for storage would be ideal.Hence we have built such a system.Once major operating systems support such a file system with adequate software,multimedia applications will move to this type of storage

# Chapter 4

# Working Platform and Environment

Due to the open source support,linux was the obvious choice of coding platform.We worked on kernel version - 2.4.20-8.The programming language was C and it was compiled using gcc compiler. Since developing a filesystem is no menial task at undergraduate level, we did not start from scratch.We analysed the coding patterns of various filesystems and their features extensively.And we have taken some of the features from existing filesystems such as FastFS,JFS and even ramfs.

## 4.1 VFS Interface

Linux provides a generic interface to work with many filesystems. This generic interface is called Virtual File System.When new filesystems are developed,it has to conform to this interface.When a file system is mounted, it registers itself with VFS.Filesystem calls from an application program is directed to the VFS.It then invokes the relevant functions associated with each filesystem.The following picture illustrates how various filesystems work transparently in linux.VFS presents a uniform interface to various filesystems under it.

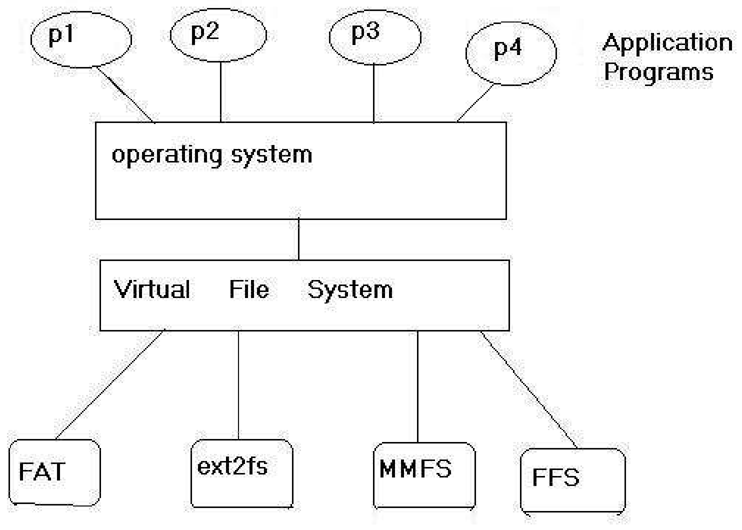


Figure 4.1: VFS Operation

## 4.2 Coding directories and methodology

In the linux kernel source directory,filesystem codes reside in the fs directory.There all filesystems have individual directories which contain the source code each one.Similarly we made a directory for MMFS and the main coding was done inside that directory.Associated changes were made in relevant make files and dependencies were added. Hereafter all the pathnames we specify are relative to the "/usr/src/linux-2.4.20-8/" directory."Config.in" file in the fs directory was modified to provide a graphical configuration interface for the filesystem.It was compiled as a kernel loadable module.When a request for the MMFS module is requested by the kernel,it gets loaded in the kernel.

### 4.2.1 Formatting Support

For formatting a partition as MMFS, we took the source code of mkfs and added MMFS support to it.The configuration settings were changed to get mkfs compiled with MMFS support. The command line for formatting a partition is

```
./mkfs.mmfs /dev/hda[number]
```

### 4.2.2 Mounting Support

Once a partition is formatted with a particular filesystem,next process is providing mounting support. The command line for mounting MMFS is

```
mount -t mmfs /dev/hda[number] /mnt/[directory name]
```

When this command is issued, operating system searches in the modules directory and loads the "mmfs" module into the kernel.In this process, MMFS registers itself with the VFS.Thereafter, all the filesystem calls to it gets routed by VFS to the function pointers registered in it. After that if you type "lsmod" you can see mmfs as a loaded module.

# Chapter 5

# Design And Implementation

Here we will examine the major features of this filesystem,its performance enhancing features,coding patterns etc.
The entire filesystem was designed keeping primarily three objectives in mind.
1. Increase the block size to a bare minimum level desirable for multimedia applications.
2. Keep maximum number of inodes in main memory,so as to reduce the disk accesses looking for inodes.
3. Keep the searching mechanism of the filesystem as simple as possible, since complicated search mechanisms invariably increase the time for access.

## 5.1 File system structure

We have set the block size at 4KB. A higher block size would have been desirable,but the VFS interface fixes the upper limit of file size at 4KB.The inode size is fixed at 64 bytes , compared to the 128 bytes of ext2 filesystem.
The Filesystem has a superblock which contains all important information about the underlying structures.

### 5.1.1 Super Block Structure

Super block contains number of inodes,number of zones,inode bitmap blocks,zone bitmap blocks,starting datazone disk address etc which are calculated dynamically based on the size of partition to be formatted. This is one of the important information which gets registered in the VFS, when the filesystem is mounted.This structure is defined in "include/linux/mmfs-fs.h".

### 5.1.2 Inode Structure

Inode contains user ID,group ID,number of links,last access time, modification time,and then data zone addresses.From inodes we can get the actual block addresses of data.From our experience,the real reason behind the fast performance of a filesystem is keeping as much of inodes in main memory.Inodes contain block addresses of seven direct blocks and three single indirect blocks.This structure is also defined in "inlude/linux/mmfs-fs.h" Inode functions are defined in "fs/mmfs/inode.c"

### 5.1.3 Zones

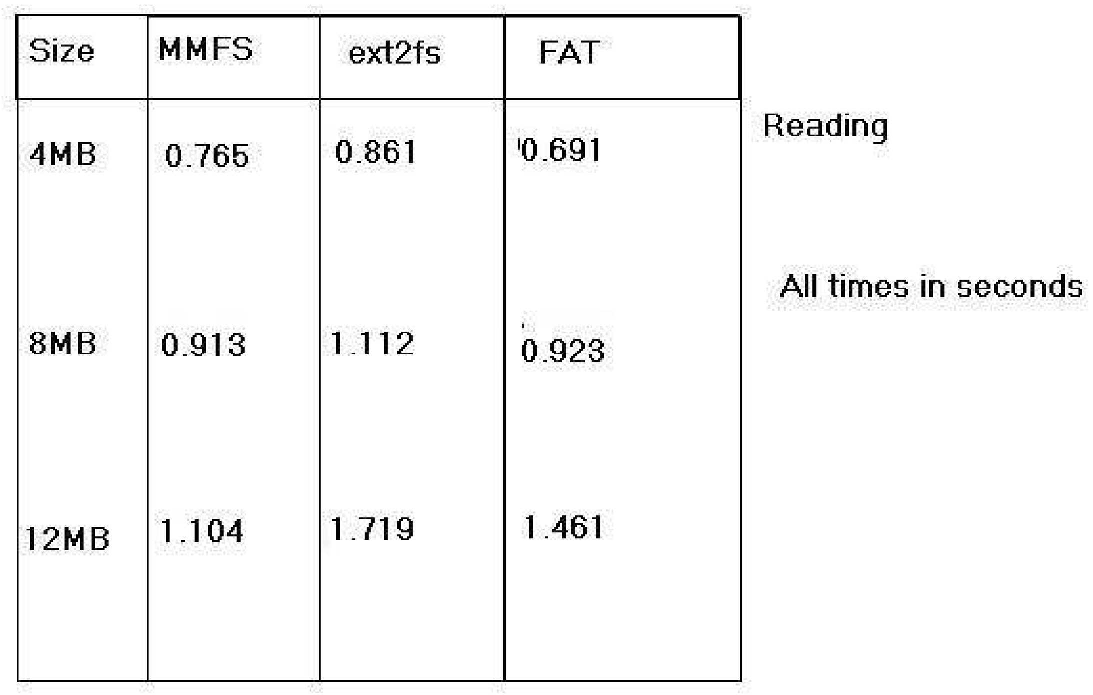
To improve seek time performance of hard disk; disk space is allocated in groups of blocks,if possible.i.e, allocation will be 8 blocks,4 blocks,2 blocks or 1 block as the case may be.These group of blocks are called zones.Through this allocation mechanism, in most cases,disk blocks of a file will lie in physically adjacent storage locations.

### 5.1.4 Inode,Zone and Block Bit Maps

To keep track of usage of inodes and zones we keep two bitmap datastructures.i.e, a bit will be set to one if the corresponding inode, zone or block is in use.The checking,setting and clearing of this is much faster than other mechanisms like free lists used for similar purpose in other filesystems.The tradeoff for increased speed in wasted space. Always a fixed portion of filesystem is allocated for these structures. Bitmap functions are deined in "fs/mmfs/bitmap.c"
The layout of the filesystem is in the following order :
superblock, bitmap structures,inodes and datablocks.

## 5.2 Performance Results

A series of observations were made comparing performance of MMFS,ext2fs and FAT in reading writing and deleting. The results are shown in the table below

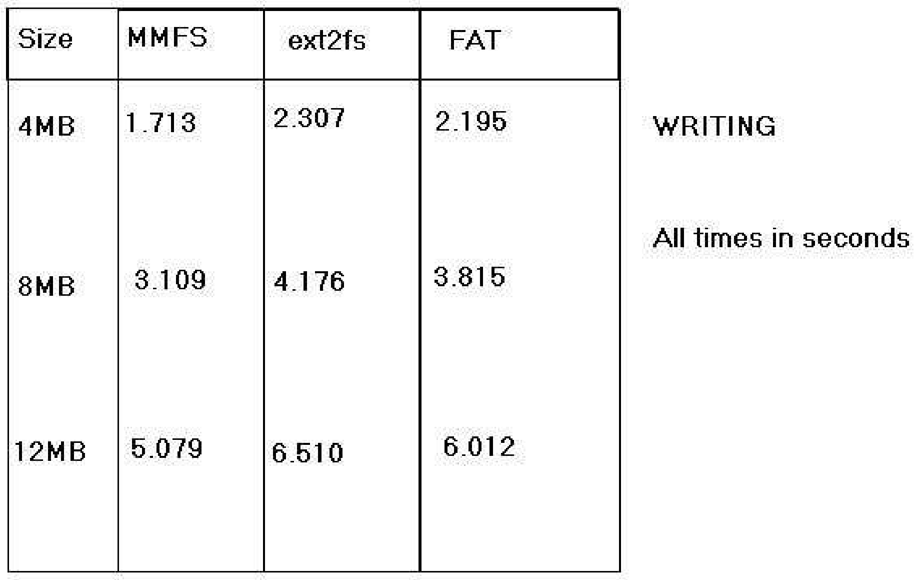


Reading

All times in seconds

| Size | MMFS | ext2fs | FAT |
|---|---|---|---|
| 4MB | 0.765 | 0.861 | '0.691 |
| 8MB | 0.913 | 1.112 | 0.923 |
| 12MB | 1.104 | 1.719 | 1.461 |

Figure 5.1: Benchmark Results for Reading

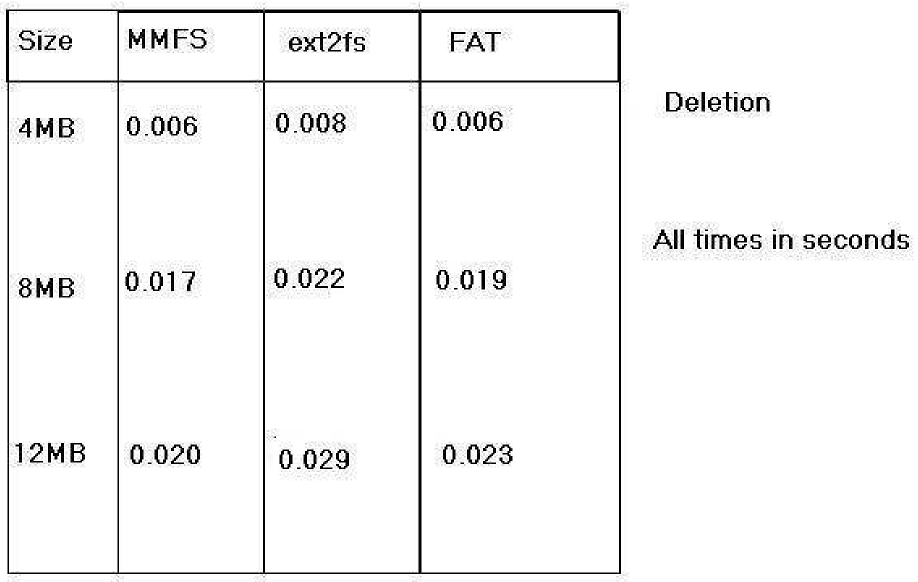


| Size | MMFS | ext2fs | FAT |
|---|---|---|---|
| 4MB | 1.713 | 2.307 | 2.195 |
| 8MB | 3.109 | 4.176 | 3.815 |
| 12MB | 5.079 | 6.510 | 6.012 |

WRITING

All times in seconds

Figure 5.2: Benchmark Results for Writing



| Size | MMFS | ext2fs | FAT |
|---|---|---|---|
| 4MB | 0.006 | 0.008 | 0.006 |
| 8MB | 0.017 | 0.022 | 0.019 |
| 12MB | 0.020 | 0.029 | 0.023 |

Deletion

All times in seconds

Figure 5.3: Benchmark Results for Deletion

## 5.3 How Is The Increased Speed Achieved ?

The increased speed for multimedia files are achieved due to the following reasons.

### 5.3.1 Simple File Operations

As evident from the allocation and retrieval mechanism follow simple lookup approach.This approach will backfire, if used in for small sized files.It is because,due to the fragmentation aspect it will be inefficient to locate them through simple lookup mechanisms. In such cases,data structures such as B-Trees,B+Trees or Dancing Trees(Of Reiserfs fame) must be implemented.

### 5.3.2 Inodes In Main Memory

Since the disk block size is 4KB and the inode size is 64 bytes, a disk block can hold 64 inodes.So once an inode disk block is read, sixty four inodes are cached by the VFS in the main memory.This is a very large number compared to the 8 - 30 inodes per block (depending upon the variables set before compilation) of ext2fs. In addition to this,we make use of spatial locality and read other inode blocks too initially.Through these measures fast access to inodes is ensured, resulting eventually in faster disk access.

### 5.3.3 Increased Direct reference

Our MMFS inode has seven direct block addresses.Each of this pointing to a 4KB block.And we have an assured 64 such inodes in memory. That counts to 1792 KB of data whose address is available without another disk reference.And there is an increased chance that single indirect blocks might be present in the main memory too.Thus ,the operating system is able to address about 2MB of data without any more disk access.

# Chapter 6

# Limitations And Bugs

Through this project we have reached a crude prototype of a product whose variations could be used by the industry after some years.In the present form it is far short of a marketable product.

## 6.1 Limitations

1.MMFS at present does not have any error recovery mechanisms,which is a must for any industrially viable filesystem.The chances of data getting corrupt after frequent operations are somewhat high compared to other filesystems.
2.Even though we aimed to make MMFS totally platform independent, it is programmed by standing within the constraints imposed by VFS and it is not tested outside linux environment.
3.Since it is residing under VFS,it is not possible for an application program to take advantage of MMFS directly. Therefore, to realise the full potential of MMFS,either one has to devise any method of interacting directly with MMFS,bypassing VFS or port it to another platform like Windows or Macintosh.

## 6.2 Bugs

Here we will look into some of the unresolved bugs in the code.These bugs are not serious to affect the performance of the filesystem,but it is essentially a source of irritation. They are:
1.The unmounting part of the filesystem is not bug free. We are getting some errors at this point. We believe that this is one of the reason behind high chances of data corruption.
2.We haven't tested this system for large movies or streaming applications.It is essential check and debug this aspect before further development

# Chapter 7

# Further Possible Works

In addition to those mentioned under limitations,here we provide general directions to people who are willing to work in this area.

## 7.1 Adding Multimedia Functionalities

Normally multimedia functionalities like JPEG,MPEG encoding/decoding exist as separate programs in the operating system.These can be integrated into the MMFS.We did attempt this unsuccessfuly.If this step is achieved, then it will be a significant milestone in the development of a complete Multimedia FileSystem.

## 7.2 Variable Block Size

Small files,if stored in MMFS cause a lot of wastage of space by internal fragmentation.This problem can be solved if one can add variable block size support to MMFS.Variable block size filesystem is really a challenge to implement.Although FAT and some of its variants have this property by a small factor, a true variable block sized filesystem is yet to emerge in computing industry.

## 7.3 Developing MMFS Outside Linux

As we have already mentioned,we were chained by the constraints imposed by VFS throughout the coding.This gave us only limited flexibility. And we believe that true performance from hardware can be extracted only by assembly language programming.So if someone is proficient in assembly language programming and have enough confidence,they can try this stuff in assembly language.We would like to add here that it will be a daunting task.

# Chapter 8

# Conclusion

Inspite of all the limitations and bugs present in the filesystem, it is running quite well and giving good performance,especially in writing to the filesystem.With its simple design and customizable interface, it can be ported to other platforms as well.We can confidently state here we have achieved the objectives of this project.It is up to the coming batches to work upon and improve the MMFS architecture.

# Bibliography

[1] "http://tldp.org/HOWTO/Filesystems-HOWTO.html"

[2] "http://www.mnl.cs.stonybrook.edu/project/mmfs/"

[3] P.Rangan,H.Vin and S.Ramanathan,"Designing an on-demand multimedia service",IEEE Communication Magazine,30(7):56-65,July 1992

[4] Y.Won,"Handling Mixed Workloads in Multimedia Filesystem", Technical Report,Division of Electrical and Computer Engineering,HanyangUniversity,Seoul,South Korea.

[5] "http://kernelnewbies.org"

[6] "http://linux-hq.com"

[7] "http://www.embedded.com"