Designing of a Virtual File System

Prasad Dumbre[1],KiranGhuge[2]

Student, Department of E & TC
SKNCOE, Pune, India
[1] *prasaddumbre98@gmail.com*
[2]*kghuge49@gmail.com*

Abstract

-In every operating System there is File system which is used to perform various operation on file like read, write, and modify which are performed as per the requirement of the user. Every OS has its own File System. There are different types of File System which are used by the different OS's. Main File Systems which are used by different OS are NTFS, FAT, EXT 2, EXT3, EXT4, JMS, XFS.In this paper similar tasks will be implemented virtually. This paper will design a Virtual File System like the File System Which is used by Linux. C language is used for writing code indesigning. Eclipse software will also be used. 20 MB RAM is required to run the code, so it is like a Virtual File System.

Keywords-*partial dynamic reconfiguration, virtual file system, abstraction.*

I. INTRODUCTION

In order to visit the temporary files frequently, Linux OS provides virtual memory file system. It can make files store in the virtual Memory. The virtual memory uses RAM also uses swap partition to store data, from the user's view of accessing files, it is not different between storing files on virtual memory and storing on hard disks. Using VFS can solve the problems of operating files directly in memory. But because of the capacity of physical memory, also when the computer power off, VFS will disappear automatically. VFS is a temporary file system, when the file has changed, the corresponding disk files without being changed. Due to the physical memory size, VFS cannot guarantee all files stored in physical memory, users need to decide which file should exist in the physical memory

II. LITERATURE SURVEY

TABLE I. COMPARISON OF REFERENCE PAPERS

Title and Author	About	Advantage	Disadvantage	Outcome
A Virtual Memory File System Based on tmpfs by Hao Li, YongpingXiong and Jian Ma	Presents the design and implementation of a file functions using Linear data structure	low cost and operating speed is very fast.	Hard to performed	Overtmpfs has all the performanceparametersoftmpfs, and also keeps pace with hard disk File system.
The Analysis and Design of Linux File System Based on Computer Forensic by CHEN Wei, LIU Chun-mei.	Mining and analyzing the useful data of the Linux operating system as it isImportant means and research directions of computer forensic	Operation speed of file parsing is high.	Complex algorithm.	Author has analyzed and designed Linux file system on computer forensic using data structures and file system parsing.

	analysis using Data structures and file system parsing.			
Analysis and Implementation of NTFS File System Based on Computer Forensics by Zhang Kai, Cheng En, GaoQinquan.	Here, the aim is to analyze and implement NTFS File System Based on Computer Forensics using NTFS file system, and file parsing system.	High speed file processing.	Expensive, complex design.	The result of analysis is displayed in a Friendly interface. A reliable data source for the computer forensics is provided.
Generic Virtual File systems for Reconfigurable Devices. By Benjamin Krill, Abbes Amira, Hassan Rabah	Aim of this paper is to develop and implement generic virtual file systems for reconfigurable devices using Generic virtual file system.	High speed file processing	Expensive, complex design.	The complete framework is implemented as a kernel module without any modifications on thekernelcode which enables the file system to supportnewkernelreleases.
XFS: A Wide Area Mass Storage File System by Randolph Y. Wang, Thomas E. Anderson.	Author has used XFS to achieve better performance availability than current generation network file systemsrun in the wide area. For this use of Integration ofmultilevel storage is done	High speed file processing.	Expensive, complex design.	XFS minimizes cache coherence stateinformation by exploiting the hierarchical nature of file system name spaceandhierarchical nature of the cluster-based organization. By integrating multiple levels ofstorage in a uniform manner

III. DESIGN

Virtual file system consists of file storage and shared library. File storage is responsible for the management of files in the virtual memory. File processing process calls the Shared library to read or write files from the VFS. In order to ensure that the data of VFS storages in physical memory instead of swap partition, considering the memory limit of 32 bits (32 bits Operating System can only access 4 G physical memory when there is not memory expansion). File loading can assume that is real-time, specific loading time is decided by file processing process.

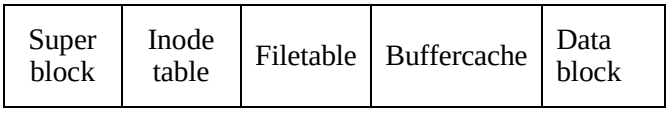


Fig.1 Structure of VFS

Some fields of superblock are,

- Block size: size of the block
- Inode_bitmap: inode bitmap table
- Data bitmap:data bitmap table
- Inodetable:inode table

- First_data_block: the location of the first block of the data area
- Data count: the number of free data blocks
- Free_inode_count:the number of free inodes
- Inode root, inode head, inode tail: LRU and Replacement algorithm use

The superblock: It is the container for high-level metadata about a file system. The superblock is a structure that exists on disk (actually, multiple places on disk for redundancy) and also in memory. It provides the basis for dealing with the on-disk file system, as it defines the file system's managing parameters (for example, total number of blocks, free blocks, root index node).

On disk, the superblock provides information to the kernel on the structure of the file system on disk. In memory, the superblock provides the necessary information and state to manage the active (mounted) file system. Because Linux supports multiple concurrent file systems mounted at the same time, each superblock structure is maintained in a list (superblocks, defined in./Linux/fs/super.c, with the structure defined in /Linux/include/fs/fs.h).

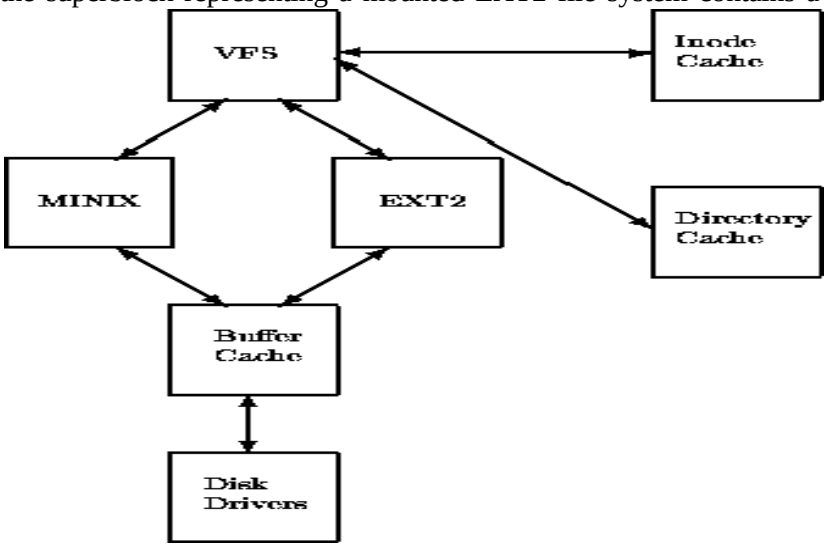
The index node:

Linux manages all objects in a file system through an object called an inode (short for index node). An inode can refer to a file or a directory or a symbolic link to another object. Note that because files are used to represent other types of objects, such as devices or memory, inodes are used to represent them also.

Note that the inode I refer to here is the layer inode (in-memory inode). Each file system also includes an inode that lives on disk and provides details about the object specific to the particular file system.

VFS inodes are allocated using the slab allocator (from the inode cache; see resources on the right for a link to more information on the slab allocator). The inode consists of data and operations that describe the inode, its contents, and the variety of operations that are possible on it. Figure 4 is a simple illustration of a VFS inode consisting of a number of lists, one of which refers to the dentries that refer to this inode. Object-level metadata is included here, consisting of the familiar manipulation times (create time, access time, modify time), as are the owner and permission data (group-id, user-id, and permissions). The inode refers to the file operations that are possible on it, most of which map directly to the system-call interfaces (for example, open, read, write, and flush). There is also a reference to inode-specific operations (create, lookup, link, mkdir, and so on). Finally, there's a structure to manage the actual data for the object that is represented by an address space object. An address space object is an object that manages the various pages for the inode within the page cache. The address space object is used to manage the pages for a file and also for mapping file sections into individual process address. Space object comes with its own set of operations (write page, read page, release page, and so on).

The VFS keeps a list of the mounted file systems in the system together with their VFS superblocks. Each VFS superblock contains information and pointers to routines that perform particular functions. So, for example, the superblock representing a mounted EXT2 file system contains a pointer to the



EXT2 specific inode reading routine. This EXT2 inode read routine, like all of the file system specific inode read routines, fills out the fields in a VFS inode.Each VFS superblock contains a pointer to the first VFS inode on the file system. For the root file system, this is the inode that represents the ``/'' directory. This mapping of information is very efficient for the EXT2 file system but moderately less so for other file systems.Internal working of below system calls:

Fig.2 Logical diagram of virtual file system

Open:The open system call can be used to open an existing file or to create a new file if it does not exist already. How it works find existing file on disk, create file tables Set first unused file descriptor to point to file table entry,return file descriptor, use absolute path begin with "/" when you are not working in the same directory of file. Use relative path which is only file name with extension, when you are working in same directory of file.

Close: Tells the operating system you are done with a file descriptor and Close the file which pointed by fd. How it works & Destroy file table entry referenced by element 'fd' of file descriptor table.

Read:From the file indicated by the file descriptor 'fd', the read() function reads 'cnt' bytes of input into the memory area indicated by buffer. A successful read() updates the access time for the file. Buffer needs to point to a valid memory location with length not smaller than the specified size because of overflow, 'fd' should be a valid file descriptor returned from open() to perform read operation because if 'fd' is NULL then read should generate error. 'Cnt' is the requested number of bytes read, while the return value is the actual number of bytes read. Also, sometimes read system call should read less bytes than 'cnt'.

Write: Write means write to a file descriptor. This is the primary way to output data from a program by directly using a system call. The destination is identified by a numeric code. The data to be written, for instance a piece of text, is defined by a pointer and a size, given in number of bytes.

Lseek:Lseek is a system call that is used to change the location of the read/write pointer of a file descriptor. The location can be set either in absolute or relative terms.

Stat: Stat system call is a system call in Linux to check the status of a file such as to check when the file accessed. The stat() system call actually returns file attributes. The file attributes of an inode are basically returned by stat() function. An inode contains the metadata of the file. An inode contains: the

type of the file, the size of the file, when the file was accessed modified and deleted, and the path of the file, the user ID and links of the file, and physical address of file content.

Chmod: In Unix-like operating systems, the 'chmod' command is used to change the access mode of a file. The references are used to distinguish the users to whom the permissions apply.

Unlink: unlink deletes a name from the file system. If that name was the last link to a file and no processes have the file open the file is deleted and the space it was using is made availablefor reuse. If the name was the last link to a file but any processes still have the file open the file will remain in existence until the last file descriptor referring to it is closed.

IV. EXPECTED RESULT

Once the complete system will be developed, a Virtual File System like the File System of LINUX OS will be functional. We will provide some functionality to our system like Linux FS. But this FS runs on RAM, so it is renamed as a Virtual File System. This virtual file system will perform the operation on different function and call the particular function after giving a specific command using singly linear linked list.

V. ADVANTAGES

VFS handle the all type of files and it performs the operation on different functions & calls the particular function after giving a specific command using singly linear linked list. So, it improves the speed of file system

VI. CONCLUSION

The VFS implementation meets its design goals. VFS has all the performance of Linux file system. And also keeps pace with hard disk file system. Application process can load the file which meets the demand advance. Especially when there are a lot of files need to be dealt with, one process is responsible for loading files, another process is responsible for the operation of files. Because the file operations are completely using the memory, improving performance. When closing a file, if the file happens changed, then VFS will write the file back to hard disk once, improving the I/O efficiency.

REFERENCES

[1] Robert A.Gingell, Joseph P.Moran, and William, A.Shannon, "Virtual memory architecture in SunOS", Proceedings of the Summer 1987 Usenix Technical Conference, Usenix Association,Phonex Arizona, USA, June 1987.

[2] Peter Snyder, "tmpfs: A Virtual Memory File System",EUUG Conference ,the Autumn 1990

[3] Wolfgang Mauerer, "Professional Linux Kernel Architecture", 2010.06, pp.583–594. Received May 17, 2018, accepted June 16, 2018, date of publication June 27, 2018, date of current version July 30, 2018.

[4] Digital Object Identifier 10.1109/ACCESS.2018.2851192 Vanguard: A Cache-Level Sensitive File Integrity Monitoring System in Virtual Machine Environment

[5] A New Design of In-Memory File System Based on File Virtual Address Framework Edwin H.-M. Sha, Senior Member, IEEE, Xianzhang Chen, Qingfeng Zhuge, Member, IEEE, Liang Shi, Member, IEEE, and Weiwen Jiang

[6] S. Oikawa, "Integrating memory management with a file system on a non-volatile main memory system," in Proc. 28th Annu. ACM Symp. Applied Computing, 2013, pp. 1589–1594.

[7] Generic Virtual Filesystems for Reconfigurable Devices Benjamin KrillNIBEC University of Ulster, Jordanstown Campus, BT37 0QB, Northern Ireland