ADDING NTFS SUPPORT TO ANDROID OPERATING SYSTEM VIA KERNEL

G V Garje[1], Tanmay Bhagwat[2], Abhishek Kale[3], Shishir Kulkarni[4], Mugdha Patil[5]

[1]HOD, Department Of Computer Engineering, PVG's College of Engineering and Technology,
Savitribai Phule Pune University, Pune, India.
[2]Savitribai Phule Pune University, PVG's College of Engineering and Technology, Pune, India.
[3]Savitribai Phule Pune University, PVG's College of Engineering and Technology, Pune, India.
[4]Savitribai Phule Pune University, PVG's College of Engineering and Technology,
Pune, India.
[5] Savitribai Phule Pune University, PVG's College of Engineering and Technology, Pune, India.

ABSTRACT:

Android is a nascent mobile operating system based on the Linux kernel that is developed by Google but lacks support for disks partitioned with the NTFS. Currently in the Google Play Store there are some third party applications to provide NTFS support to Android OS. But the main disadvantage of such applications is that the phone needs to be rooted (flashed) which may be tedious for a lay user and in many cases may void the device's warranty. And above all, a custom ROM needs to be installed to avail the NTFS functionality.

This paper covers a technique to provide native support for NTFS (New Technology File System) in the Android operating system by avoiding the rooting process. Hence to use NTFS-SD card or NTFS HDD we won't require any third party applications to perform basic read/write operations. This is implemented with the help of the Virtual Filesystem layer (VFS) which is a subsystem of the kernel that implements the file and the file system-related interfaces provided to user-space programs. The NTFS will rely on the VFS layer to enable programs to use standard UNIX system calls to make basic operations like read/write as well as advanced features which NTFS has.

KEYWORDS:

$MFT, $LOGFILE, DENTRY, FAT, LKM,INODE, NTFS, SUPERBLOCK, VFS.

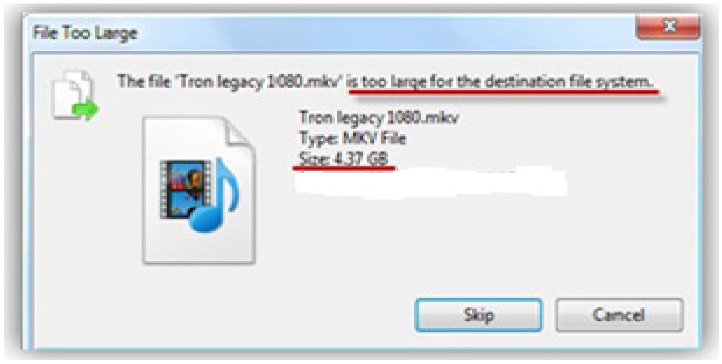
1. INTRODUCTION

A. The Android Scenario

As of year 2012, approximately 500 million Android devices had been activated with 1.3 million activations per day. In early 2013, at Google I/O event, they announced that there had been 900 million Android device activations setting a new record. Today Android has the largest installed base of any mobile operating system and above all its devices also sell more than Windows and IOS devices combined. An important aspect of Android operating system; inheritance from Linux has made it possible not only to be used in mobile phones, but also in smart TVs, smart cars and other electronic equipment too. Thus the scope of this OS is very wide. Initially at the introductory level it was sufficient to have FAT system support to Android, but now horizons of Android have been extended and hence there rises a necessity to add other widely used file system (like NTFS) support to Android kernel to make it more pragmatic.[7]

Presently, only a few Android devices(less than 1%) have full NTFS support (read/write) because it is not enabled by default in any Android kernels. By default Android kernel provides support for FAT12, 16, 32, Ext3 or Ext4 file system. Most Android devices include a micro SD card slots and can identify, register and read/write data from/into these micro-SD cards formatted with FAT12, 16, 32, Ext3 or Ext4 file systems only. High-capacity storage media such as USB flash drives and USB HDDs of NTFS file system are not supported at all. [7]

Generally the FAT32 storage is handled by a Linux Kernel VFAT driver, while 3rdparty applications are required to extend this support to other popular file systems such as NTFS, HFS Plus and ex-FAT.


File Too Large

The file 'Tron legacy 1080.mkv' is too large for the destination file system.

Tron legacy 1080.mkv
Type: MKV File
Size: 4.37 GB

Fig. 1: FAT file system.

The image above clears the limitations of the FAT as it clearly says 'too large for destination file system'.

B. NTFS Features that we have implemented at a glance:

These will be the features that are included in our implementation:

Fault tolerance: All modifications in NTFS are recorded in a log file called $LogFile .When system crashes, NTFS examines the logfile and restores the system to a consistent state with minimal data loss.

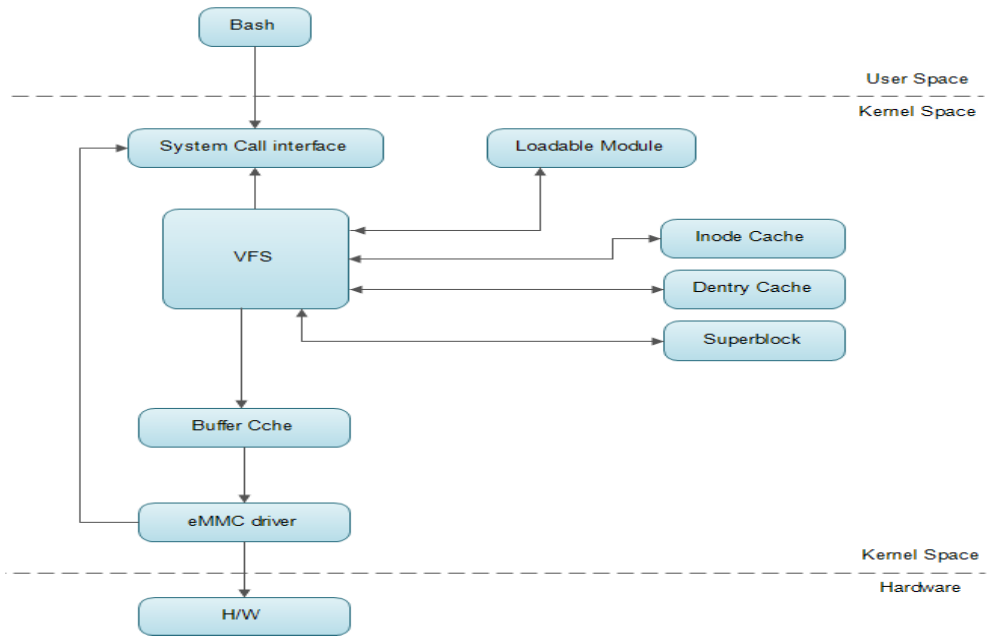
16 bit Unicode: It is a predefined set of characters for naming various files and folders. It provides support for various languages used across the world like English, Russian, Hebrew, etc.

Storage Management: NTFS doesn't depend on the underlying system's hardware. Furthermore there's no limit on the file size that it can store

Journaling: NTFS can natively compress and encrypt files without the use of a separate application. The file is encrypted/compressed during the file write, and decrypted/uncompressed during the file read operation. All this compression/encryption is transparent to user.

I. Basic System Architecture

Fig. 2: System Architecture.



Kernel Space and user Space: Android kernel is monolithic kernel and therefore the file system is implemented in kernel space. The kernel provides some system calls by which the file system modules can be registered with the kernel.

VFS: VFS is an abstraction layer that specifies an interface between kernel and a concrete file system. It is because of VFS that it is possible to separate low level file system from the kernel.
Inode Cache: The kernel keeps track of the file system using in-core inodes that are index nodes. The inode cache keeps information of such inodes to avoid multiple addressing of inodes.

Dentry Cache: Dentry is required for maintaining the hierarchical structure of the file system and it represents a specific component of the path. It provides front end to the inode cache. Dentry cache thus, keeps track of all these dentries.

Superblock: Contains all the global information of the file system.

2. REGISTERING FILE SYSTEM

A module implementing a file system type must announce its presence only then can it be used. Its task is divided into subtasks as follows: (i) to have a name and identify it, (ii) to know how it is mounted, (iii) to know how to search for certain files and traverse through the directories, (iv) to know how to find (read, write) file contents.

Android’s file systems can be built as modules and these modules can be ‘demand loaded’ as they are needed or loaded by explicitly by using insmod. A file system module has to register itself with the kernel whenever it is loaded and similarly unregister itself while unloading. Thus each file system's initialization routine registers itself with the Virtual File System and is represented by a ‘file_system_type’ data structure which contains the name of the file system and contains a pointer to its VFS superblock read routine. These ‘file_system_type’ data structures are put into a list pointed at by the file systems pointer. [1][2][3][4] Each ‘file_system_type’ data structure contains the following information:

Superblock read routine:
This routine is invoked by the VFS when an instance of the file system is mounted.

File System name:
The name of the particular file system. For e.g.: Ext or NTFS or FAT

3. MOUNTING THE FILE SYSTEM

The mount system call as the name suggests attaches a file system to the big file hierarchy at some indicated point which may be specified by the user or to some default point. Parameters needed include: (i) a device that has the file system (HDD, SD-card, etc.) (ii) A particular directory where the file system on that device must be attached to, (iii) lastly the type of file system. [1][2][3][4]

II. Main modules

1. Loadable Kernel Module

- Loadable Kernel Module is nothing but a piece of code that can be loaded or unloaded into the kernel on demand as and when required
- Used to extend functionality of kernel without even needing to reboot the entire system
- Without modules, we would've to build monolithic kernels which would be inefficient
- Linux kernel in spite of being a monolithic kernel it supports kernel modules making it modular

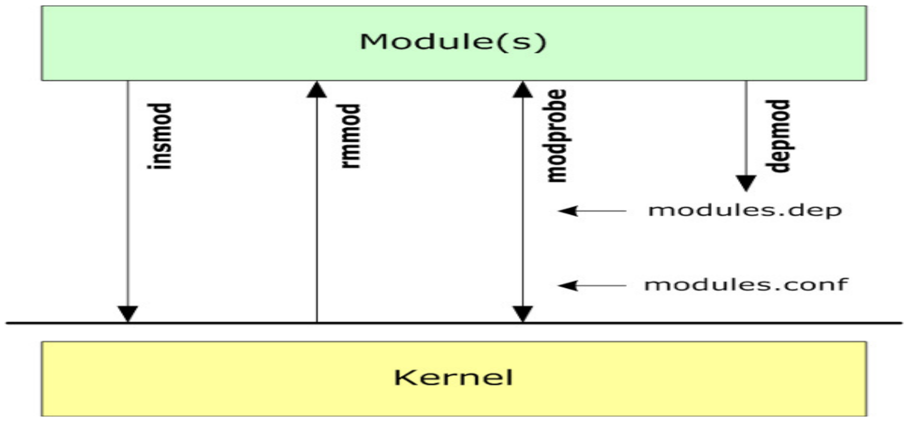


Fig. 3: Kernel Module Basic Operations

2. VFS objects

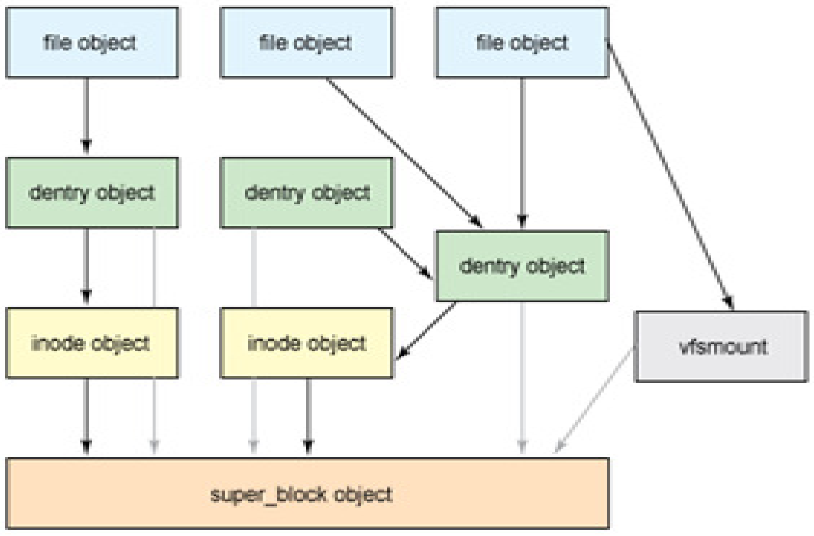


Fig. 4: VFS Objects and their relationship

2.1. Superblock

- It is a data structure that corresponds to the file system superblock or the control block of the file system, which is stored generally in a special sector of the disk
- Superblock is
- It can be read directly from disk for disk based file system or can be created on the fly for non-disk based file systems as and when required
- The 'alloc_super ()' function is used for creation and initialization of superblock objects
- The most important item in the superblock object is s_op. It is a pointer to the superblock operations table.
- Each entity in this particular structure acts as a pointer to certain functions that are used to make changes to the superblock
- In short, the superblock contains all the global information of the file system[1][2][3]

2.2. Inode

- The index node or the inode as commonly known is a data structure used to represent a file system object which can be either a file or a directory.
- The inode object is used to point out and make available all the information which will be needed by the kernel in order to manipulate a particular file or a directory
- These inodes are generally nothing but pointers to a inode table which will contain all the actual data of a file or a directory
- The number of inodes can be limited, thus limiting the number of files on the file system.
- If the file systems does not have inodes, in such cases the file system must obtain the information from wherever the related information is stored on the disk
- The inode object is represented by a structure called inode and is defined in 'linux/fs.h' file[1][2][3]

2.3. Dentry

In Android OS just like the LINUX OS, the VFS treats all directories as some special files or simply files. In the path /tan/t1, both tan and t1 are files tan being the special directory file and t1 being a regular file. An inode object can represent both these components. Now many a times the VFS needs to perform directory-specific operations, such as path name lookup. Now this path name lookup involves translating each component of a particular path, to check its validity, and checking its flow to the next component.

Thus, this is where the VFS uses the concept of a directory entry called the dentry. A dentry is a specific component in a path. To sum up resolution of a path and traversing its components is a substantive task, time-consuming and full of multiple string comparisons. Using the dentry object we make the whole process time efficient as well as easy. The VFS is able to construct dentry objects as and when required, when performing various directory operations. Dentry objects are represented by structure 'dentry' and defined in 'linux/dcache.h' file. [1][2][3]

2.4 File

- File object represents a file opened by a certain process which will be in the memory
- This file object is created in response to an open() system call and destroyed in response to a close() system call
- Multiple file objects can co-exist as multiple processes can co-exist at a time
- The object points back to dentry that actually represent the open file
- In short the file object represents an inode together with a current (reading/writing) offset.

3. Struct mount VFS and fs_struct

A structure used for mounting is 'vfsmount' which contains all the parameters necessary for successful mounting. Its definition can be found in 'mount.h' file. A secondary structure called 'fs_struct' will determine how to interpret the pathnames which are referred to by a particular process. E.g.: The typical reference is current->fs whose definition can be found in a file called 'fs_struct.h' [4][5][6]

4. Plug and play NTFS device

Thus following the above procedure any ARM or non-ARM device will able to register, mount and successfully perform basic read/write operations on a NTFS device and any NTFS drive can be used as a plug and play drive..

3. CONCLUSION

One completion any Android device will be able to read/write and make all basic file operations provided by NTFS. All this will be achieved by making changes in the kernel and not through any 3rd party application.

References

[1] Robert Love, “Linux Kernel Development”, Pearson Education, 2nd Edition, 2005.
[2] “Downloading and Building Kernel”, http://source.android.com/source/building.html
[3] “Interactive Map of the Linux Kernel”, http://www.makelinux.net/kernel_map
[4] Tim Jones,”Anatomy of Linux File System”, http://www.ibm.com/developerworks/linux/l-linux-fs
[5] “The Linux kernel archives”, http://www.kernel.org
[6] “The Unix heritage”,http://minnie.cs.adfa.edu.au/TUHS
[7] “Android(Operating System)”, http://en.wikipedia.org/wiki/Android_%28operating_system%29