Application based File System (ABFS)

Krishna Modi
B. Tech. (comp.) 3rd Year
Mukesh Patel School of Technology Management & Engineering
JVPD Scheme Bhaktivedanta swami Marg
Vile Parle (w), Mumbai- 400 056

Prof. Prashasti Kanikar
Assistant Professor
Mukesh Patel School of Technology Management & Engineering
JVPD Scheme Bhaktivedanta swami Marg
Vile Parle (w), Mumbai- 400 056

ABSTRACT

A file system is designed to store the data into the storage device efficiently and without any data loss. Every data stored on the storage device is used by some or the other application. When an application is opened, 90% probability is that the user will open a file which is supported by that application. But the files in the storage devices are not stored in a sorted manner. They are generally stored on the first come first serve basis. We have paid special attention and made efforts to design a file system which stores the data in a sorted manner so that while accessing the data in the file system, the disk header has to make least possible movement thus reducing the seek time and giving an optimal response time. Software is responsible to make the hardware perform its work efficiently. This has influenced the design of ABFS which stands for Application Based File System.

General Terms

File System, File storage method, Data Management, Sorted Storage Mechanism.

Keywords

Application Based File System, File System, Sorted File System, Journaling, Reduced Seek Time.

1. INTRODUCTION

In ABFS, the data is sorted according to the application which uses that data. Every data is accessed by some application. In ABFS, every application has its own database of files. Every file in ABFS is referred as an item, and a folder is referred as group. A group is a collection of items organized as separate entity by the user or system to refer items with specific similarity by a common entity. Every application data table maintains a record of the items and the groups. These items are highly classified according to their applications. Whenever an application is opened, the files supported by that application are only accessed, which are actually stored together in ABFS. This actually helps in reducing the time to search for data, hence reducing the seek time of the disk.

The ABFS is highly comfortable for operating systems meant for personal computers. An operating system needs to have a complete different architecture to implement ABFS because the complete design of ABFS is a new revolution in the history of file systems.

2. RELATED WORKS

Microsoft designed its file system popularly known as File Allocation Table (FAT) in 1980[1]. This file system had a very basic design with all the data entries stored in a simple tabular format. FAT was initially designed for Floppy disks which later went in use for Windows operating system up till Windows 98. It is still highly preferred and recommended file system for flash drives and other small portable storage devices.

FAT32 is the latest version of FAT file system introduced in 1996 which supported up to 2 Terabyte of maximum volume size. It provides no sorted storage or any security measures against data loss. In this paper, we have revamped the design of FAT with more features and better architecture maintaining the same tabular format for recording entries of data items stored in the storage device.

3. PROPOSED DESIGN

Every application installed in ABFS has its own Application Data Table (ADT). This ADT contains all the entries of the items and groups stored by that application in the system. Hence, every item is stored and recorded in a sorted manner. ABFS makes it easy for the operating system to uninstall an application from the system. When an application is uninstalled, all the data related to that application can be easily calculated and removed from the system for complete removal of the application. This removes the redundant or useless data storage in the storage device. Some applications are inbuilt by the operating system and they cannot be uninstalled. This secures the data or items stored in that application data table from getting deleted even accidently.

For every Application Data Table (ADT), its application is referred as native application and other applications are referred as external applications, and their ADT as External Application Data Table. The items in the ADT have limited access; they can access items only under their level. The Application Based File System has no limitation for memory addressing. It can address huge amount of memory locations and clusters because of its ‘Repeat and conquer’ approach. According to it, the ABFS can have an item of maximum size 128TB. It can address a volume of maximum size 12.5PB which is sufficient for any file system present currently [2].

The efficient storage technique removes the redundant data from the storage device and uses the storage space efficiently and wisely. As data of same format as stored together, the head seek time is minimum. Whenever an application is opened, the operating system reads the location of that application’s ADT and informs the head to start reading for data from the specified location. The disk read/write head starts reading the data from where the Native ADT for that application is stored. Hence the response time on accessing any file in the file system is very less and optimal as compared to any other file system. This is supposed to be the only file system which stores the data in a sorted manner without much rewriting, shifting the data from one memory location to another, or using defragmenters.

The Application Based File System also allows the applications to add their flags as per their needs. Every application can use set four flags in its own data items for its own use. The parameters or settings stored and needed by the application can be stored in the file system itself, applications do not need to create any file to store these data and encrypt them.

4. FEATURES (Proposed)

4.1 Sparse Files

Sparse files are files which contain sparse data sets, which are files with segments stored at different file offsets with no actual storage space used for the space between segments. When a file is read back, the file system driver returns zeros for any data that does not actually exist, so the file may appear to be mostly filled with zeros. For instance, Database applications sometimes use sparse files. Because of this, we have implemented support for efficient storage of sparse files by allowing an application to specify regions of empty (zero) data. An application that reads a sparse file reads it in the normal manner with the file system calculating what data should be returned based upon the file offset. As with compressed files, the actual sizes of sparse files are not taken into account when determining quota limits.

4.2 Quotas

In ABFS, the files are sorted according to the applications using them and their formats/extensions. A rough idea can be easily drawn from the ADT how much memory to be saved for a specific application. Hence, a Quota is maintained for every application to separate every data of different applications from each other. This maintains the uniformity in the stored data. A quota is an imaginary amount of memory allocated for an application, though an application can exceed its quota limits without any complexity or special permissions.

4.3 Single Instance Storage

When there are several groups that have different, but similar, items, some of these items may have identical content. Single instance storage allows identical items to be merged to one item and create references to that merged item. SIS consists of a file system filter that manages copies, modification and merges to items; and a user space service that searches for items that are identical and need merging. SIS was mainly designed for remote installation servers as these may have multiple installation images that contain many identical files [3]; SIS allows these to be consolidated; changes to one copy of an item will leave others unaltered. This is similar to copy-on-write, which is a technique by which memory copying is not really done until one copy is modified.

4.4 Atomicity

ABFS is a Journaling File System. It maintains a record of all the changes to be in the journal before actually committing them to the main file system. This helps in achieving atomicity for the File System operations. If any copy, paste, modification updating or new file creation process is executed; the journal table keeps track of the changes, and then applies them on the main File System. If the process was successful, only then is the change made in the main File System, otherwise not. The "all or nothing" rule of atomicity is followed by the ABFS. For the journaling, a journal table is maintained by the file system. On every bunch of successful executions, the journal table is checked for the changes made and updates them to the main file system. The journaling file system was basically developed by IBM Developers [4]. It is utilized to achieve atomicity in Application Based File System.

4.5 No Journaling Mode

Journaling maintains the integrity of the file system by keeping a log of the currently ongoing disk changes. However, it is known to have a small overhead [5]. Some people with special requirements and workloads can run without a journal and its integrity advantages. In ABFS the journaling feature can be disabled, which provides a small performance improvement.

4.6 Access Level Limitations

In Application Based File System, a tree structure is formed. Index is the source node, having the several applications as its children nodes. Every application has its items and groups as children nodes; every group has its own children nodes having items or subgroups in them. Every node has a limitation that it can access only other nodes at level same as of itself which are under the same parent node. Under this condition, every item in an ADT won’t be able to access any other item in EADT of any External Application. This is known as access level limitation. Every item has access to a limited number of items in the file system. If the operating system is affected by virus, it won’t be able to affect much of the files in the file system unless it is in the system files or has gained abrupt access permissions. Access Level Limitations add up to the security feature of the file system. There is no chance that the virus enters the system files because they cannot be accessed by even the administrator.

4.7 Restricted System Files

The system files in the Application Based File System are restricted to be accessed by even the administrator. Only the root user can read the system files, still he cannot modify/update them or write a new file in the system files. The system files comprises of all the files of the operating system. The operating system files are not useful for the user. They are just used by the system. It is not efficient to display the system files to the users. User should be able to access only the data which is useful. No system data should be viewed by the users. For better user experience and security, the system files are restricted for user access. System files are not visible for any user except the root user. This is also useful for security against viruses. A virus in the system files can access any file in the file system as the system files and folders have complete permission for all items in the file system.

4.8 External Application Access

Every application can have its own data table. Some applications may require access to the EADT of External Applications. For example, let the operating system provide a music player application called as Music Box. Let there be a third party application called VLC media player. This application needs access to all media files which are under the ADT of its native application. This is possible in ABFS. If the any of the data type used by both the application is same, then the items of that data type or extension can be access by the External Application from the ADT of the Native Application. But the External Application can just access the EADT. To modify, update or write in it, the External Application needs to have permission for it.

4.9 Sorted Storage

The items in the ADT of applications are stored at one place. Every ADT of application is stored at a distance from every other ADT of another application because of using the quotas. Every ADT has its own space and items of same format/extension are stored nearby. Hence, when an application starts, all the items needed by the application are available at one place and the disk head doesn't have to seek to different locations for accessing the items. As a result, the data is accessed easily and at a faster speed. The Sorted Storage technique is achieved by the ABFS in a very simple manner and carried out smoothly.

4.10 Repeat and Conquer Approach

The Application Based File System has crossed the limitations faced by other file systems in an extra ordinary way. The ABFS can have an item of maximum 128TB in size. This File System can address 12.5PB of memory locations in a storage device. 1PB is equal to 1024TB of memory. This is all possible because of the 'Repeat and conquer' approach of the ABFS. In this approach, ABFS repeats itself and covers the entire storage space. If every cluster is equal to 64 sectors and every sector is equal to 512bytes of memory [6], then with an address of 32-bit, the memory that can be addressed is 128TB ((2^32)*64*512). After every 128TB of memory, the ABFS file system index table is repeated and the next memory location is counted from 0 again. To address this file system, an object oriented approach is used. Every memory location can be addressed via its file system index table. There can be 100 index tables, starting from 00 to 99. To access location 3 from 2^{nd} index table, we would write the memory location as: 02->#00000000000000000000000000000010. The index table is the main table which contains all the systems files, user records, application records and table locations and journal table. This is the main file system table and is repeated to address the large amount of memory locations.

5. PARAMETERS

5.1 Flags

5.1.1 Read only

If this flag is set for any item or group, then that item or group can be just read by any external application. No external application can modify/update such items or groups even if they have special permissions. For native applications, this items or groups are completely accessible, and can be modified in their native application.

5.1.2 Read-write Protection

If this flag is set for any item or group, that item or group is neither readable nor writable by any external application even on any special permission. Only native application can read and write such items or groups.

5.1.3 Hidden

Any item with hidden flag set is not visible to any external as well as native application. This flag can be unset only by the native application, not the external application.

5.1.4 Administrator only

If this flag is set, the item is not allowed to be read or written by any of the application, let it be external or native. Only administrator or anyone with administrative privilege or permission can access that item. This flag can be unset only by administrator via the native application only.

5.1.5 Item

If the data stored in the Application Data Table is an item, then this flag is set.

5.1.6 Group

If the data stored in the Application Data Table is a group, then this flag is set.

5.1.7 Stable

If an item is modified / updated and is saved then this flag is set. If the modification in the item is not saved then this flag is unset.

5.1.8 State

If an item is currently in use by some application then this flag is set, otherwise it is unset. It is necessary for achieving Integrity of data items.

Table 1. Flags

R O	**R/W**	**H**	**A O**	**Item**	**Group**
Stable	**State**	**A B**	**A B**	**A B**	**A B**

The Applications have rights to add more flags to the data items. Four flags are reserved for applications for their use in every item. This gives the applications more control over the data they have access. The application developers can use this flags more efficiently than using more memory space for writing a new item to maintain a record of properties of these data items.

5.2 Time Stamp

Two time stamps are stored for every item in the ADT. The date and time of creation and last access is saved for every item and group.

5.3 Item Name

The maximum number of characters that can be used for naming an item is 64. As observed, mostly up to 10 characters are widely used for naming any file. Giving more amount of memory to store name of items seems useless and inefficient use of memory. So, 64 characters are allocated for names of each item and 3 characters are allotted for the extension of the item.

6. DISK STRUCTURE

The Application Based File System contains an Index table. This table consists of all the system files, user records, application names and table locations, journal table. The Index table is the main data entry table which contains all the system files. The system files are the operating system files and records.

This system files are followed by the user records. In this, all the user details and records are maintained including access permissions, user id, etc. Then the application details are entered in the Index table. In the application details, the name of application and the location of Application Data Table are stored. To uninstall an application, the system has to just delete the entry of an application from this table. The memory allocated for the application automatically gets rewritten when a new application is installed.

After the Application details comes the last but not the least, Journal table in the Index table. This gives the location to the Journal table in the memory location. The Journal table is used to store the details of all the operations that are being committed on the file system. If the operation is complete, the journal records it and attempts to make changes in the main file system. If it is successful, then the journal record is completed and de-allocated for new record. This table is used to achieve atomicity in file system operations. The Index table looks like below table:

Table 2. Index Table

System files	**#00.....110**
User records	**#00.....131**
Application 1	**#00.....632**
Application 2	**#00.....733**
Application n	**#23.....234**
Journal Table	**#45.....342**

The Application Data Table has the details of all the items used and stored by that application. Every ADT is secured. There are basic details about every ADT, such as Application name, User access, Data Types, EADT and Write permitted. Here, Application name is the name of the application whose ADT is referred. User access contains the user id of all the users who are permitted to access the ADT on special conditions. Data types are the extensions that are used and acknowledged by the application. This data type entry is useful to access the ADT of external applications. If the data type of another ADT matches the data type of one ADT, then that application can access the items of EADT of that specific data type. The write permitted parameter contains names of all the applications which are permitted to read and write into the ADT of this application. These are the basic structure details of every ADT. The ADT contains entries of all items in it and the groups of items in it. Every address in the ABFS is of 32-bit length giving 2^32 addresses available for addressing the volume.

7. COMPARISON

As per our aim, we have made our best attempt to design a file system which is sufficient and efficient for the computers of this era and is better than other file systems currently used. ABFS can allocate a file of maximum size 128TB whereas for Ext4, NTFS and FAT it is just 16TB, 16TB and 4GB respectively. The implementation of ABFS is easy as compared to NTFS and Ext4. NTFS sorts the data according to their sizes whereas in ABFS, the data is sorted according to its size as well as according to its type and format. ABFS is supposedly the only file system which can reduce the effects of virus because of its secured structure which is not possible in any other file system known up till now. It is a Journaling file system but it also provides the No Journaling mode like in Ext4 [7]. ABFS uses a new type of data storage structure which we call as Application Data Tables. This structural design of ABFS makes it more strong and resilient against any attack still keeping it simple enough for the programmers and developers simple enough to understand and implement.

All these features of ABFS compared with the other file systems can be clearly seen in the following table.

Table 3. Comparison Table

Property	**FAT**	**NTFS**	**EXT4**	**ABFS**
Maximum File size	4 GB	16 TB	16 TB	128 TB
Implementation	Very Simple	Simple	Complicated	Simple
Sorted Storage	No	Yes	No	Highly Sorted
Virus Protection	No	No	No	Yes
Journaling	No	Yes	Yes	Yes
No Journaling Mode	No	No	Yes	Yes
Reduced Seek time	No	Yes	No	Yes
Storage Structure	Table	Sorted Table	Tree	Application Table

8. CONCLUSION AND FUTURE SCOPE

The Application Based File system overcomes the basic limitations faced by various file systems. It has no limitation to maximum memory volume. ABFS gives better security features with access levels and permission based control over the file system. It provides a great solution for the problems of virus and worms by limiting the access to just its group. No redundant data is stored in the storage drive and the stored data is automatically sorted. Lesser seek time for disk head and increased response time for optimal results is another advantage of ABFS. This file system is scalable and fault tolerant. It is robust in nature. A file cannot be reached by an application can be reached via another application using the EADT access permission for same data types. No unwanted data is displayed to the user. The system files are safely stored in the system section of the index table.

High efficiency of data storage and retrieval is accompanied in this file system. It achieves atomicity for each of its file operations. Separate access permission for different users can be easily granted. The applications have greater control over the data stored in the file system.

This file system is highly effective for operating systems of all kinds. All mobile operating systems, tablet computer operating systems and laptop, desktop operating systems can implement this file system. This file system is not useful for external storage drives.

Hence, we have successfully designed a file system to overcome the drawbacks of currently used file systems and implement all the security features, efficient techniques of data storage. We look forward to better future of this file system for using it in the highly efficient operating systems.

The design of ABFS can be modified to suit several situations, platforms and environments. It can be suitably modified to work in all possible environments. There is yet much development to be made in the design to make ABFS suitable for networking. In the next phase of research, we would implement the design of ABFS with more improvements and advancements.

9. ACKNOWLEDGMENTS

We thank Prof. Sudeep Thepde for his guidance and help all over the project for detailed study of file systems. His interest towards the subject and good understanding capabilities helped us to be more devoted to the subject and complete this paper. We sincerely thank Dr. D J Shah for his supervision and constant motivation to write this paper in more versatile manner.

10. REFERENCES

[1] "Microsoft Extensible Firmware Initiative FAT32 File System Specification, FAT: General Overview of On-Disk Format". Microsoft. 2000-12-06.

[2] Silberschatz, Abraham; Galvin, Peter Baer; Gagne, Greg (2004). "Storage Management". Operating System Concepts (7th Ed.).

[3] "Single Instance Storage in Windows 2000". Microsoft Research and Balder Technology Group. August 2000.

[4] "Model-Based Failure Analysis of Journaling File Systems", Vijayan Prabhakaran, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau

[5] Jones, M Tim (2008-06-04), Anatomy of Linux journaling file systems, IBM Developer Works, retrieved 2009-04-13

[6] Silberschatz, Galvin, Gagne. Operating System Concepts, Sixth Edition. John Wiley & Sons, Inc.

[7] "The new ext4 filesystem: current status and future plans", Avantika Mathur, Mingming Cao, Suparna Bhattacharya, June 27th–30th, 2007, Ottawa, Ontario, Canada