File System Implementation using FAT

Rushikesh Nalla (ID:201401106)* and Omkar Damle (ID:201401114)†
IT-308 Operating Systems course, DA-IICT

A filesystem consists of methods and data structures that an operating system uses to keep track of files on a disk or partition i.e. the way the files are organized on the disk. File system implementations vary in design. We have used FAT file system which has a file allocation table located at the beginning of a logical volume. We simulate a virtual disk using a simple file. The data rates of the disk are measured for varying block sizes.

INTRODUCTION

We have implemented a FAT based file system using a virtual disk. This virtual disk is a single file stored on the real file system. We used C++ for implementation. The main aim of this project was to understand and implement the basic functionalities of a FAT file system. The functionalities have been described in detail in subsequent sections. The key points of our project are :

- We have designed an interface which can be used to execute all basic file system functions defined in the report.
- Our file system can be used to read and write large files which use multiple data blocks. A data block has a size of 2kB. A file of size, say, 6kB will use 3 data blocks. The data blocks need not be contiguous in memory.
- We have run a performance test to find the average data rates for varying block sizes.

Implementing a virtual disk

A disk is made up of surfaces, tracks and sectors. We will consider the disk that consists of a number of sectors which are numbered consecutively starting from 0 to a pre-defined maximum. These sectors can be termed as a collection of blocks that form a linear, numbered list. All the blocks have same size. We have considered that each allocation unit is made up of a single block.

Structure of the file system

File systems are stored on disks. Most disks can be divided up into one or more partitions, with independent file systems on each partition. Sector 0 of the disk is called the MBR(Master Boot Record) and is used to boot the computer. The end of the MBR contains the partition table. This table gives the starting and ending addresses of each partition. The first thing the MBR program does is locate the active partition, read in its first block called the boot block and execute it. FAT stands for File Allocation Table. File allocation involves dividing the disk into blocks - units used for allocation and the FAT describes which blocks are used by which les.

The layout of the FAT based file system is as follows:

- The first section is the super block which contains:
  - The start of FAT table
  - The start of Directory table
  - The start of Data Area
  - The number of files
- The next section is the FAT structure which contains an integer value of the next block number that follows it in the file to which it belongs. -1 is stored if it is the end of file and -2 if it an empty block.
- Next is the directory table which contains the file names and meta-data(their starting blocks, length and its validity status) for describing files.
- This is followed by the data area which contains the data associated with the files. It contains the rest of the blocks on the disk.

IMPLEMENTATION DETAILS

The virtual disk has been implemented as follows:

- Total size of the virtual disk is 21504KB which is approximately 21MB.
- Size of each block is 2048 bytes.
- Total number of blocks are 10752.
- The first block is the super block.
- The FAT table has an integer value stored in it so the total size needed for 10752 blocks is 10752*4 bytes and we divide it by the block size which is 2048 bytes. So we need 20 blocks for the FAT table.

- The file descriptor table is used to store file descriptors. Each entry in the file descriptor table corresponds to an open file. Each entry contains filename, permission mode, validity status, and a buffer containing the currently accessed block. Maximum of 32 blocks can be open at any given point of time.

The disk is actually a single file and data is stored in the form of blocks so we need functions to access the disk i.e. to read and write data block-wise. The following functions are used for creating and accessing the disk.

- make-disk: function to create and empty, virtual file disk
- open-disk: function to open the disk file
- close-disk: function to close a previously opened disk file
- block-write: function to write a block of data to the disk
- block-read: function to read a block of data from the disk

The functions for managing the file system are as follows:

- fFormat: function to create a new file system and initialize meta-data associated with the same.
- assignTables: function to mount the file system on the disk. It opens the disk and loads the metadata of the file system thus making it ready to use.
- saveTables: function to unmount the file system from the disk. All the metadata of the file system must be written back to reect all the changes made to the file system. Then the disk should be closed.

The following are functions to access and modify files stored on the file system:

- fOpen: function which takes the file name and permission as input to open a file and returns a le descriptor which can then be used to subsequently access the file and -1 on failure.
- fClose: function which takes file descriptor as input to close a file. The corresponding le descriptor is closed. It returns 1 on success and -1 on failure.
- fCreate: function which takes file name as an input to create a new empty file in the table directory. The file has to be opened to access it later.It returns 1 on success and 0 on failure.
- fRemove: function which takes file name as an input to delete a file from the directory table. The blocks containing the metadata and the data of the file are freed. It returns 1 on success and 0 on failure.
- fRead: function which takes file descriptor, pointer to buffer and size as input to read data (in bytes) from the file into a buffer. It returns 1 on success and 0 on failure. Note that if multiple blocks of data are to be read, then the function follows the links using the FAT table.
- fWrite: function which takes file descriptor, pointer to buffer and size as input to write data (in bytes) into the file from a buffer. It returns 1 on success and 0 on failure. Note that if multiple blocks of data are to be written, then the function follows the links using the FAT table. If new blocks are required, a strategy similar to first fit is used.
- fRename: function which takes the old and new name as input to rename the file. It returns 1 on success and 0 on failure.
- fList: function used to list all the file names in the root directory.

BLOCK ALLOCATION METHODS

There are different methods for allocating blocks in memory. There are some advantages and disadvantages for both.

Contiguous Allocation

Each file is assigned requested number of successive blocks in memory.

- Each file on creation can ask for a certain number of blocks. The file cannot expand in memory.
- If sufficient free contiguous blocks are not available then the file will not be created.
- Since the blocks are physically adjacent to each other, the access time improves as compared to non contiguous allocation.

Disadvantages:

- This method leads to internal fragmentation of memory. A file may not fully utilise all the memory blocks allocated to it and there will be a lot of scattered free blocks in memory.
- If sufficient free blocks of memory are available, a file may not be created since the blocks are not contiguous. This leads to external fragmentation.

Non Contiguous Allocation

We need an allocation technique which is dynamic in nature i.e. allows growing and shrinking of files. This is taken care of by non contiguous allocation. In this methods the blocks of the file are linked together with each block containing pointer to the next block. The most primitive type of non contiguous allocation is using a linked list to store pointer to next block in file. This method eliminates external fragmentation, but suffers from increased number of block accesses. This method suffers heavily for random access of data. Instead, we can use a FAT table. Each block has a corresponding entry in the FAT table. If the block is a part of a file, then the FAT entry corresponds to pointer to next block. If the block is the last block in the file, the entry is -1. Free blocks have FAT entries are denoted by -2. Thus in order to access any data in file, we can traverse through the FAT table, find the pointer to the required block on disk and read the block. Only one disk access is required.

Advantages:

- This does not cause external fragmentation because all the blocks can be used as per the memory request for each file.
- It improves disk space allocation as every block can be used even if they are not successive.

Disadvantages:

- We need to perform a sequential traversal through all block entries in the File Allocation Table till the required block.
- There is an overhead for storing pointers to the next block in memory.
- Error in storing a pointer can make subsequent blocks of memory inaccessible.

Inode

Inode is a data structure that describes a file or a directory. The first section consists of meta data like permissions, owner ID, Size of file, time last accessed, time last modified etc. The next part of the structure consists of 12 pointers to the data blocks. These are used for fast data access when the file size is small. Then comes the indirect access blocks where a pointer to a table is stored which contains pointers to rest of the data blocks. This helps in storing vast amount of data as there is another layer. If this is not enough then we can use two layers of interaction and this continues as required. It allows for random access of data as we have pointers to each block stored in a table and also it prevents external fragmentation. This is another way in which files data can be kept track of but we have not used this structure for our implementation.

PERFORMANCE ANALYSIS

We compared the data rates for different block sizes. Refer to fig 2 and fig 1. If the block size is small, then a greater number of disk accesses are required to fetch the data. However, if the block sizes are large, smaller number of disk accesses are required. As a result, the data rates increase for larger block size. On the flip side, the internal fragmentation within a block increases with larger block size. Hence the block size needs to be optimally chosen to balance the trade off between data rates and memory utilisation.

Block size(in bytes)	Average Time required(in nano sec)	Data size (in MB)	Data rate(in gigabits/sec)
512	36000000	2	0.44
1024	20000000	2	0.792
2048	14000000	2	1.131
4096	13000000	2	1.218
8192	11000000	2	1.439
16384	10000000	2	1.583

FIG. 1. Observed data rates for different block sizes

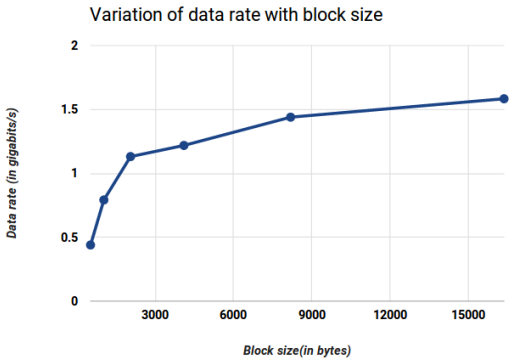

FIG. 2. Effect of block size on data rates

LEARNINGS FROM THE PROJECT

- We learnt about disks and disk partitions. Each partition can have its own file system. A Master Block record keeps track of the partitions on the disk and the pointers to the partitions.
- This project provided us a means of applying theoretical knowledge regarding file systems in practice.

The FAT is a very popular file system which was used in hard disks and floppy disks extensively in the late 20th century. The FAT file system is useful for linear access of data, because one needs to travel sequentially through all the blocks before reaching the desired one.

- We also learnt about how directories are arranged in a file system. A directory contains the block pointers to the first block of every file. In order to form an hierarchy of directories, we can model a directory as a special file.

- We learnt the concept of mounting of file system of a disk - the process in which the meta data associated with the file system is loaded.

- We compared different allocation methods like contiguous allocation, linked list allocation, FAT and inode.

---

* 201401106@daiict.ac.in
† 201401114@daiict.ac.in

[1] Andrew S Tanenbaum, Modern Operating Systems, 2nd edition. Morgan Kauffman. Prentice Hall