File System Tools – Design

Pushpa Devi	Mangesh Thakare	Rajasimhan Baskar
A20246561	A20250279	A20276195

1. Introduction

This document encloses the design details of project 3 on MINIX 3 operating system. An important requirement for the project is

1. To develop a tool fileinfo that when invoked from the shell with a file name argument, it will report the following:
   - The list of all processes that have this file opened
   - The list of blocks on disk that contain this file
2. To develop command filefrag that will determine the amount of external and internal fragmentation in the file system.

2. Design

2.1 FILEINFO command

The fileinfo command is a simple application which determines the process numbers who have opened the file and the block numbers where the file is stored on the physical media.

2.1.1 FILEINFO – The list of PID's that have this file open.

This subroutine determines the vnode number corresponding to the file using eat_path(). Vnode structure contains the inode number of the file and the file systems endpoint number. It loops through the list of fproc array, which is an array of process. Fproc structure has an array of file descriptor entries. For each file descriptor entry which is a pointer to the filp entry in the filp table, it checks if the filp entry's vnode pointer and given file's vnode pointer are same to determine if the process has opened the file.

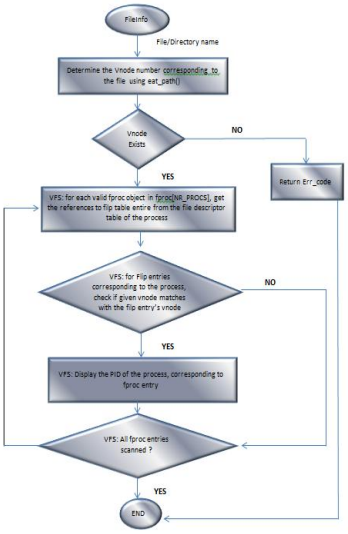


Fig1: code flow of the fileinfo() system call for displaying the list of pid's that has opened the file

2.1.2 FILEINFO – block numbers of blocks occupied by the file on disk.

The system call internally finds the virtual node corresponding to the file. The VFS portion of the fileinfo() sends a request REQ_GETBLOCKS to the MFS server process passing the inode number of the file. Number of blocks occupied by a file are calculated by the MFS server process. MFS reads the inode corresponding to the inode number of the file passed by the VFS. Inode contains the total file size and the size of the one block, from these two information total number of blocks is calculated. For each of the i_zone array of inode

a. For each direct zone, iterate through the blocks of this zone and display the block numbers. For each block iterated, decrement block_count. If block_count reaches zero, it indicates end of file has been reached.
b. If end of file is not reached still, get the Single indirect block. For each direct zone in this single indirect block, iterate through the blocks of this zone and display the block numbers. For each block iterated, decrement block_count. If block_count reaches zero, it indicates end of file has been reached.
c. If end of file is not reached still, get the double indirect block. From double indirect block, get the next level of single indirect blocks. For each direct zone in this single indirect block, iterate through the blocks of this zone and display the block numbers. For each block iterated, decrement *block_count.* If *block_count* reaches zero, it indicates end of file has been reached.

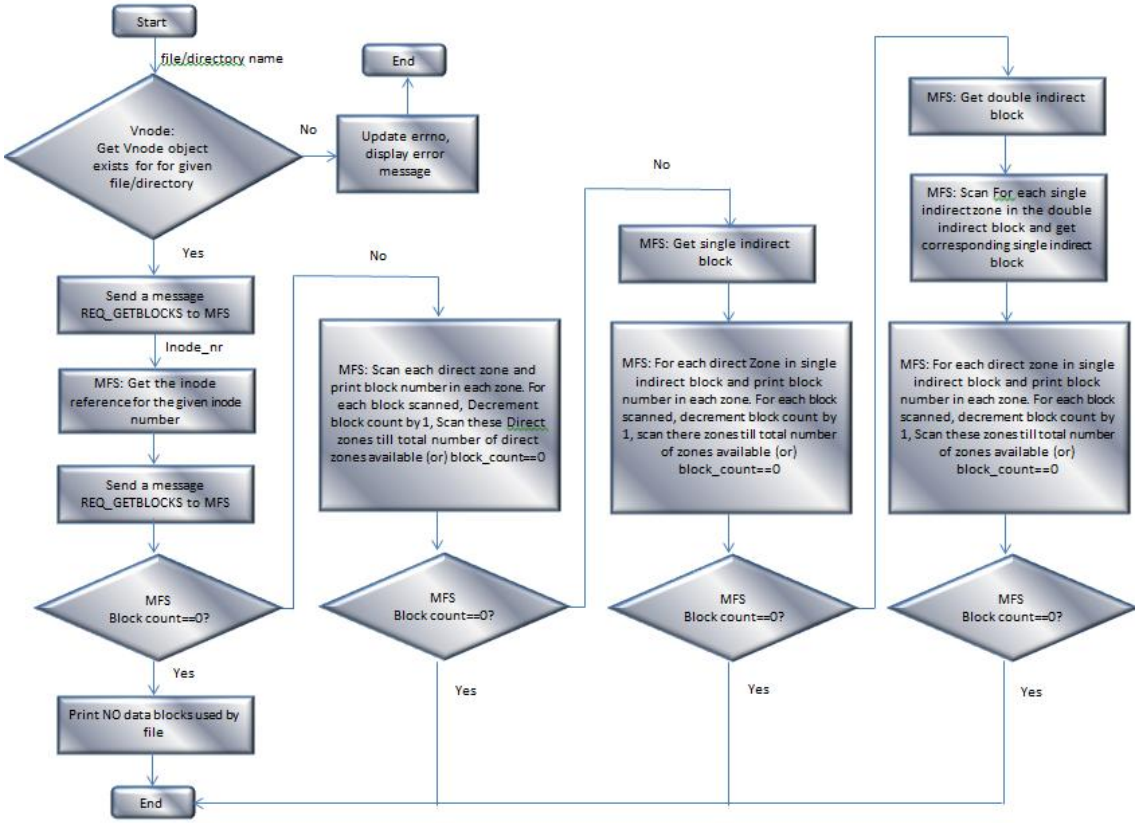


Fig2: code flow of the fileinfo() system call for displaying the list of pid's that has opened the file

2.2 FileFrag – External and Internal Fragmentation.

2.2.1 The externfrag determines the amount of external fragmentation in a file system.

The system call internally finds the virtual node for each file and the directory. The VFS portion of the fraginfo () sends a request REQ_EXTFRAG to the MFS server process passing the inode number. MFS accesses all the inodes present in the file system and finds there are any external fragment blocks in each inode. It determines the amount of external fragmentation based on total data blocks of the file system.

External fragmentation of file system = Total External fragment Blocks / Total data blocks of file system.

Total data blocks of file system = Total number of data zones multiplied by number of blocks per zone.

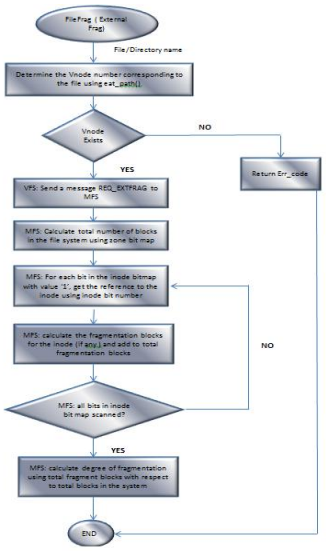
For each of the inode bitmap blocks, iterate through the bits and get the reference to the inode using inode bit number if the bit value is 1.

For inode of each file, *total blocks* used by the inode is calculated as file size / block size. *Used blocks* in the last zone of the file is calculated as total_blocks_used % blocksPerZone. Add the remaining unused blocks in the last zone to the external fragmentation

fragment_blocks = blocksPerZone - blocks_in_last_zone;

Add the fragment_blocks of all the inodes. This gives the Total Fragmentation blocks. Once all the bits in the inode bitmap is scanned, determine the amount of external fragmentation in the file system (in percentage) as

extFrag = ((float)TotalFragBlocks / TotalBlocksInFS)*100;



2.2.2 The Internal frag determines the amount of Internal fragmentation in a file system

The system call internally finds the virtual node for each file or the directory. The VFS portion of the fraginfo () sends a request REQ_INTFRAG to the MFS server process passing the inode number of the file or directory. If it's a directory then it list all the files in that directory. Each of the Inode of the file gives the pointer to the block. MFS accesses the inodes and finds there are any Internal fragmentation for a particular file.

Internal fragmentation is calculated as block size – (filesize % blocksize)

3. System Call guide (manual page)

System Call	***do_fileinfo()***
Purpose	The system call returns either the list of all process that has this file opened and/or list of blocks on disk that contain this file depending on the option chosen.
Prototype	int do_fileinfo(void)
Library Function	int fileinfo(char *fname,int mode)
Input	Message m7 of minix packed with filename and mode.
Returns	Success : 0 Failure : -1
Exception	Wrong option chosen by the user, File may not exist.
Usage	fileinfo filename [-p] [-b] • -b : Displays only the block numbers of blocks occupied by the file on disk • -p : Displays the list of pid's that has opened the file • Default option: Displays both the list of pid's that has opened this file and also the block numbers of the blocks of the file

System Call	**do_fraginfo ()**
Purpose	Determine the amount of external and internal fragmentation in the file system which includes the largest, least and average unused storage for each type of fragmentation.
Prototype	int do_fraginfo (void)
Library Function	int req_extfrag(endpoint_t fs_e)
Input	Message m7 of minix packed with type as REQ_ EXTFRAG.
Returns	Success : 0 Failure : -1
Exception	Zone number absent
Usage	fraginfo [-i] [-e] [-d] • -i: Displays Internal fragmentation details • -e: Displays External fragmentation details • -d: Displays list of all devices.

Data Structures

Following Existing Data Structures of minix are used for implementation of file system tools.

The following data structures are used to list the process Ids that has opened a given file.

1. **Struct fproc** : This structure is defined in fproc.h. fproc.h has an array of fproc objects with array size NR_PROCS. Fproc object maintains per process information. A process will occupy an entry in this array. To find all the processes that exist in the system, we have to iterate through this array.

2. **struct filp** :- This structure is defined in file.h . The filp object represents an opened file. It specifies how the file was opened, which vnode it refers to and the current file position. file.h has an array of filp objects with array size NR_FILPS. This array is called the filp table. It is an intermediary between file descriptors and vnodes. When a process opens a file, the file descriptor entry is a pointer to filp entry in this filp table. Few important fields in this structure are:-

3. **struct vnode** :- This structure is defined in vnode.h represents the inode of the file in VFS server process.

Data Structures used for Listing the blocks on disk of file and also calculating external fragmentation :

The following data objects are used to get the list of blocks on disk of a given file and determine the degree of external fragmentation.

1. **struct super_block** (in mfs/super.h):- This structure object contains the super block information of the file system.

2. **struct inode** (in mfs/inode.h):- This structure object corresponding to a file contains all the file header information like file size, access time, modification time etc and an array of zones.

Other Data Structures used for calculating external fragmentation:-

1. **Inode bitmap** :- The inode bitmap appears after the superblock in the file system. Each bit in the inode bitmap is used to indicate if inode exists, with 1 indicating inode exists and 0 indicating inode doesn't exist. Super_block contains the number of inode bitmap blocks.

2. **Zone bitmap** :- The zone bitmap appears after the inode bitmap in the file system. Each bit in the zone bitmap is used to indicate if zone exists, with 1 indicating zone exists and 0 indicating zone doesn't exist.

Exception/ Error Handling

There can be exception situation in the file system call. Following error are added to the system to report specific types of errors.

Error Message	Fileinfo (List of pids)	Fileinfo (Listof Blocks)	externfrag	InternalFrag
"File name too long"	Yes	Yes	Yes	Yes
"Invalid argument"	Yes	Yes	Yes	Yes
"No such file or directory"	Yes	Yes	Yes	Yes
"Arg list too long"	Yes	Yes	Yes	Yes
"Zone number absent"	-	Yes	Yes	Yes

Extra Credit

Fragmentation information is useful for deciding file system structure i.e block size, zone size etc. Current requirement only lists largest, least and average unused storage. This information may not be much useful. To make more sense of information we have decided to extend the scope and collect information about Number of files, Number of directories.

The fragmentation distribution provided in 8 bins is as follows: 0-511, 512 - 1023, 1024-1535, 1536-2047, 2048-2559,2560-3071,3072-3583, 3584-4095. The bin range is auto adjusted depending on size of a block. The above example is for block size 4096.

The above information states the health of your filesystem. The maximum distribution in 0-511 range implies you have done a good job minimizing bit loss. If 3584-4095 region is crowded that means the block sizes are not properly designed for the current files on your disk. You can try reducing block size and increasing zone size in corresponding proportion not to increase access time. This would improve disk utilization.