ATLANTIS PRESS
Advances in Engineering Research (AER), volume 131
3rd Annual International Conference on Electronics, Electrical Engineering and Information Science (EEEIS 2017)

Algorithm of Embedded File System in Industrial Storage Management

Hui-Zhong LIU
Institute of Applied Mathematics Hebei Academy of sciences
shijiazhuang,050081,China
E-mail: lhzh114@sohu.com

Keywords: Log-structured; Changeble-Tnodelength; Dynamic; Static

Abstract. Aiming at the NAND flash which is widely used in industrial storage management, the embedded file system named 'Jkffs' is designed, it is a log-structured file system, a direct addressing algorithm is descripted, Tnode-tree, changeble-Tnodelength are used, the method of Chunk array is mentioned also. The dynamic wear-leveling algorithm and static wear-leveling algorithm according to the different erasing times of block are presented in Jkffs to realize the garbage-collection and wear-leveling.

1. Introduction

NAND flash memory is widely used in the industrial storage management in which embedded system is the core. The log-structured file system designed according to the physical structure characteristics of NAND flash can solve the embedded system of mass storage management issues.

NAND flash array can be divided into a series of blocks. Each block contains a number of Pages, and the block is the smallest erasable unit. Erasing a block means defining all the bits as "1" (defining all bytes as "FFh"). Writing operation, through the programming, turns erased bits from "1" to "0". The unit of reading and writing is the Page, the Page contains data area and leisure area (OOB, out-of-band), and OOB area is used for software designer. Writing Page and erasing block are the basic operations of NAND flash. Programming all bytes of blocks for OXFF can release space [1]

2. Establishment of The File System

2.1. The Basic Ideas of the File System

The file system we establish is named "Jkffs"(Jk flash file system). Additional information is stored in OOB area, which is designed to realize the management of NAND flash[2]. All the components in the file system, including file, directory, links, devices etc., are separately regarded as a file .Each file has a special head page store file to preserve the head of mode, length, file name, father object labeling and other information. The basic unit for writing in NAND flash is the Page called Chunk in the file system, which has different meanings. Page refers to the actual data storage areas on the NAND flash Memory, while the Chunk is the logical data storage areas allocated by the file system. Their sizes can be different.

For example, Samsung K9K8G08U1A,With 2048+64 bytes as a Page of NAND flash chips, it has 8G capacity, the file system uses 8 bytes on OOB area to store the relative file system information, and design a erasing-times mark (Erase_num, 8 bits), to recover block and realize the wear-leveling. The directory structure of the file system must be built in the memory during the process of loading the system. Reading OOB contents is only needed to scan each Chunk and from the system marks in the OOB, it can be determined whether the Chunk is head or data. Then according to the contents of the file head Chunk and the information such as object ID, Chunk ID of data Chunk, establish a corresponding object in the memory for each file. After all the block has been scanned, it would establish the relationships of all objects, and form a kind of architecture in RAM. Thus, "Jkffs" would be loaded successfully [3].

2.2. Addressing of the File System

Efficient addressing can be realized by creating a node tree. Structure definition:

```
union jkffs_Tnode_union {
union jkffs_Tnode_union *internal[8];
                                }
```

This is a pointer array of length 8. The node tree at the bottom, created according to this structure, becomes the leaf nodes and the middle is internal node. They have the same structure. When the node is the internal node, each element of the array will point to the next layer of child node; When the node is the leaf node, the array will be splitted into 16 long integer with N bits (Tnodelength), which is the storage location of the file contents in the flash (that is, Chunkid).

The Tnodelength of leaf nodes determines the maximum addressed space for the file system. For example, Tnodelength = 16, means it can represent 2^{16} =65536 Chunk. As for a NAND flash with 2K Chunks , the largest address space c is 128 M.

It is very convenient to find the files through the node tree. Each Tnode used by the internal node has 8 pointers, so three binary codes are needed to index it, therefore when the tree grows a layer high, three codes would be added into the Chunkid. In turn, each three nonzero Chunkid represents a layer of the internal nodes. Meanwhile, each Tnode used by leaf nodes has 16 pointers, so four binary codes are needed to index it.

Only the lowest level Tnode would be established when Tnode structure started to be built, And when the number of the Chunk was more than 16, an internal Tnode would be established in the tree, and NO. 0 pointer would point to the lowest level Tnode. While more and more Chunk is read, new Tnode will be added and the node tree grows more and more high.

2.3. Changeble-Tnodelength in the Node Tree

Tnodelength of bottom node in the node tree determines the maximum space which can be addressed in “Jkffs” . If the size of the Chunk is 2048 bytes, 128MB NAND flash need 65536 Chunks, and it needs 16 bits to index Chunk. Similarly, 256MB NAND flash needs at least 17 bits to index all the Chunks and 512 MB NAND flash needs 18 bits. In order to facilitate processing, the TnodeSize must be the multiple of 32bits, TnodeSize = (Tnodelength * 16) / 8, the unit is byte, Tnodelength represents the length, the Tnode of level 0 is 16 physical Chunk index, From the foregoing, Tnodelength is the size of physical Chunk index, and the unit is bit. In order to make TnodeSize the multiple of 4 bytes, 256 M flash’s Chunk index bits has to be 18 bits at least. Thus Tnodelength = 18 can directly address 512 M. To insure the foregoing rule, Samsung K9K8G08U1A, the Tnodelength can be set 22, thus the file system has the maximum space: 2^{22} * 2K = 8G for addressing.

The way by increasing Tnodelength can realize the management of NAND flash. But such established node tree occupy too much RAM space in embedded system, which lower system performance, and it is not quite flexible for the protean design of embedded system.

In system initialization process the information of NAND flash is first read, the total number of Pages can be obtained by multiplying blocks of flash (nblocks) by Pages of each block, which means that file system need to manage these Pages, and then figure out Tnodelength of bottom node .specific algorithm is as follows:

1) Calculated the total number of Pages x = nChunksPerBlock * nBlocks
2) when X > 1, shift X 1 to right ,bits= bits+1
3) repeat , until x <= 1
4) if X is odd number,X=X+1
5) end

Flow chart as shown in figure 1

Bits is bit length, and assign to Tnodelength. This way can solve the problem that fixed Tnodelength occupies too much RAM in the system, and can manage the most kinds of NAND flash.

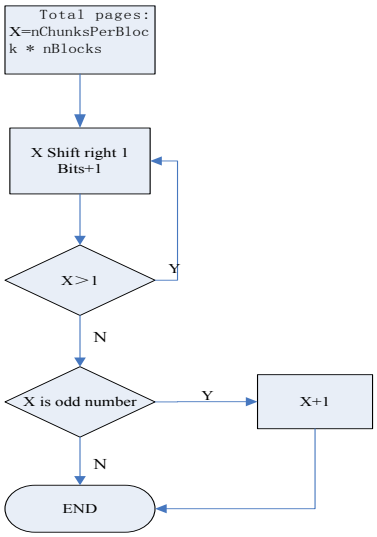

Fig. 1 Tnode-length algorithm flow chart.

In the actual system design, if bits is too large, Chunk array can be considered. Make one Chunk to match more Pages, through which can reduce the Tnodelength. Thus, the goal of managing NAND flash memory can also be achieved. That is, through synthesizing certain Chunk into an array with a same Id may also increase the addressing space. At this time, the Page and Chunk are different. Supposing that one Chunk represents two Pages and two continuous Pages can be considered one of element of d_Chunk array, including 4096 bytes data, here d_Chunk [0], d_Chunk [1] represent relative Pages separately, as shown in Figure 2:

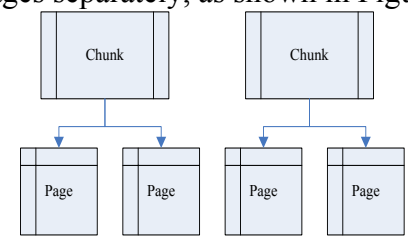

Fig. 2 Chunk array and Page relationship schemes.

3. Garbage- Collection and Wear-Leveling

The life of the NAND flash is limited, it is determined by the maximum erasing times of the block. Therefore, wear-leveling algorithm should be design to distribute the erasing and writing operation evenly on each block and its impact on performance should be as low as possible[4]. The process of erasing blocks and reuse it is called “garbage-collection”. Considering the using frequency of data with garbage-collection and wear-leveling, we designed two algorithms: dynamic and static wear-leveling algorithm.

3.1. Dynamic Wear-leveling Algorithm

The working range of dynamic wear-leveling algorithm is the updated frequently data space and unused space, it will be realized in the writing of data. Algorithm flow chart is shown in figure 3:

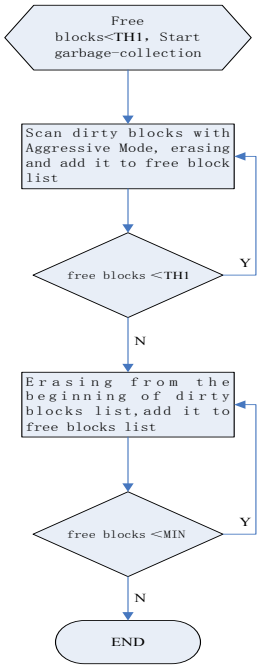

Fig.3 Garbage-collection Flow chart.

The algorithm supplies a good way to manage the dynamic data in flash memory. It can guarantee a fixed number of free blocks, as well as can solve data writing delay and accumulation. Two values were used in the block recovery algorithm above, they are Min and TH1. Min is the number of recovered dirty block each time when the idle block list is empty. Next, TH1 will be explained.

In this algorithm, garbage-collection will happen when free block list is empty, while the recovery of dirty block may need to copy the effective data of dirty block first, so the erasing should be done by descending order of useless data Page when recovering block, here TH1 means the lower limit of dirty block though "Aggressive Mode", the recovery can be operated from the beginning of the list of dirty block when the value is more than that.

The Min must meet the conditions for Min ≥ TH1. If Min is too large, the number of free blocks will increase, accordingly, the available memory space will reduce; If Min is too small, it cannot solve the problem of data accumulation completely. In "Jkffs", Min is 1/10 of the total block, namely Min = nBlocks / 10, TH1 = Min / 2.

3.2. The Realization of Static Wear-leveling

In the practical application of the embedded system, data storage has such characteristics: most files are small files, but large files take up most of the storage space; the blocks of small files are updated much frequently[5]. Thus the following situation will appear: the erasing blocks of small files have more erasing times than that of big files, which leads erasing times of small files' erasing blocks will reach the upper limit of erasing times faster and the life of the whole flash memory will be used up.

In order to deal with this kind of static read-only data, we designed static wear-leveling algorithm. When the garbage-collection was triggered, read the EraseNum from OOB of the erase block in the clean blocks list, which marked the erasing times, then find out biggest and smallest erasing times. Then do subtraction between them, if the value is bigger than threshold, it can be concluded that erase with minimum number of erasing stored static data, and it needed to be moved. And because the static data are usually bigger and take up more erase blocks, the number of blocks with the minimum time of erasing is more than 1. Therefore, in order to better achieve erase equilibrium, it is necessary to scan the clean block list again after moving operation, and find out the block with maximum erasing times. Specific algorithm is divided into the following steps:

1) Scan clean block list, and find out the blocks with the biggest and smallest erasing times, then record the number EraseMax and EraseMin respectively, and name the blocks A and B respectively
2) If "EraseMax - EraseMin≥TH2", erase block B store the static data, it must be done with moving operation.
3) select one block from the free block list, copy the data of A to it; Then erase A, and then copy the data of B to A; And then put B to the dirty block list for the storage of data updated frequently.
4) Repeat above three steps, until "EraseMax - EraseMin≥TH2" is false, That is, all the static data have been moved.

The value of TH2 is very important in the algorithm. If the threshold is too large, the number of static file storage blocks needed to be released will be greater than that of free blocks, and effective data cannot be all received. Conversely, if threshold is too small, blocks for static data file storage cannot be released promptly, and frequent data moving will reduce the performance of the system. Usually it is better to set the value of TH2 between 200 to 500. In "Jkffs", TH2 = 250.

4. Conclusion

At present the most reasonable management of NAND flash memory of the file system mostly use the log-structured ideas, such as JFFS, Yaffs, F2FS etc [6]; The file system "Jkffs" is developed and completed with the ideas, Practice has proved that "Jkffs" could flexibly manage many NAND flash with various sizes.

References

[1] NAND Flash Technical Paper, SLC-Large Block 8 GBit, 1Gx8, K9K8G08U1A, http://www.samsung.com/global/business/ semiconductor productList , 2007.

[2] T.S.Chung, D.J.Park. A survey of flash translation layers. Journal of Systems Architecture, 55:332-343, 2009.

[3] BK.Kim, DH.Lee. a log-structured B-Tree index structure for NAND flash SSDs. Design Automation for

[4] P. Desnoyers. Analytic models of SSD write performance. ACM Trans. Storage, 10(2):8:1-8:25, March 2014.

[5] B. Van. Houdt. A mean field model for a class of garbage collection algorithms in ash-based solid state drives. ACM SIGMETRICS Perform. Eval. Rev., 41(1):191-202, 2013.

[6] SH.Park, JW.Park, A Pattern Adaptive NAND Flash Memory Storage Structure. IEEE Transactions on Computers, 61(1):134-138, 2012.