Computer Standards & Interfaces 48 (2016) 80–89

Contents lists available at ScienceDirect

Computer Standards & Interfaces

journal homepage: www.elsevier.com/locate/csi

Design and implementation of split/merge operations for efficient multimedia file manipulation

Youngmin Kim [a], Youngjoo Woo [b], Hyeonsu Lee [b], Euiseong Seo [b,*]

[a] Division of IT and Mobile Communication Business, Samsung Electronics, Suwon, Republic of Korea
[b] College of ICE, Sungkyunkwan University, Suwon, Republic of Korea

ARTICLE INFO

Article history:
Received 22 December 2015
Received in revised form 6 April 2016
Accepted 26 April 2016
Available online 10 May 2016

Keywords:
File systems
Multimedia editing
Non-linear editing
Flash memory
Ext4
FAT

ABSTRACT

Various kinds of smart devices, including personal video recorders, smart TVs, smart phones, and digital cameras frequently manipulate multimedia files, such as frame trimming or inserting. However, because the standard file system APIs do not provide appropriate operations for multimedia file editing, application developers have to implement editing operations by using the conventional read and write operations. Therefore, editing multimedia files unnecessarily incur a large number of I/O operations, resulting in significant performance drawbacks. This paper proposes the design of two novel file system operations, *split*( ) and *merge*( ), to remedy the performance overhead of manipulating multimedia files. Unlike conventional file system operations, these operations are carried out only at the metadata level and are accompanied by no actual data movement, so they take negligible time to finish regardless of file size. In addition to the design proposal, this paper introduces the implementation details of the proposed design for three representative file systems: Ext4, FAT32, and exFAT. The evaluation with a commercial digital camera showed that the movie file trimming took less than 1 s for most cases, which is well over 1000% improvement over the conventional approach. In addition, the number of I/O operations, which significantly affects the lifespan of flash memory storage components, was suppressed to less than 1% of that with the conventional approach.

1. Introduction

Due to the ever-growing popularity of multimedia sharing over the Internet and social network services, users of smart consumer electronics, including smart TVs, smart cameras, personal video recorders, and smart phones are frequently recording, editing and sharing multimedia files, such as videos, music, and voice recordings via diverse Internet and social networking services. In addition, multimedia file storage via cloud systems has proliferated across the market. These services usually limit the length or size of the multimedia files that can be uploaded. Therefore, trimming unnecessary parts of multimedia files is necessary [1]. In addition, easy-to-use multimedia authoring tools, which can insert a video clip in the middle of another clip or concatenate two multimedia files, enable non-professional users to create diverse multimedia works.

Because the conventional file system operation set does not provide file operations that can insert data in the middle of an existing file or remove part of an existing file, such multimedia file manipulation has been implemented by using read and write operations only. For example, to trim an unnecessary video fragment, an application copies the necessary part to a new movie file and then deletes or backs up the original file.

This approach requires a large number of read and write operations and the movement of huge volumes of data considering the large size of multimedia files in general. Consequently, this prolongs the response time and degrades the user experience. In addition, the write amplification from copying the existing data blocks significantly degrades the lifespan of the flash memory inside the embedded storage systems due to its limited program count [2]. Last but not least, this copy operation demands additional storage capacity to accommodate the new multimedia file and original file simultaneously. Therefore, the file manipulation will fail if the amount of free storage space is insufficient.

Diverse approaches have been proposed to deal with the aforementioned inefficiency for PCs or server systems, and most of them depend on extra metadata structures. However, they are not feasible for consumer electronics because the edited files can be spontaneously transferred to the other devices via memory cards or USB connections, and the extra information required to access the edited files will spoil the compatibility with the other devices.

To resolve this issue, this paper proposes two novel primitive file-manipulating operations: *split* and *merge*. As their names imply, the merge operation concatenates two existing files to merge them into a single file, and the split operation separates an existing file into two smaller files. These operations are conducted by modifying the file systems metadata structure, which takes a negligible amount of time. Thus, neither operation produces a copy nor move operation over the data

* Corresponding author.
E-mail addresses: ymkim74@gmail.com (Y. Kim), mongmio@skku.edu (Y. Woo), hyunsu@skku.edu (H. Lee), euiseong@skku.edu (E. Seo).

http://dx.doi.org/10.1016/j.csi.2016.04.005
0920-5489/© 2016 Elsevier B.V. All rights reserved.

blocks. Consequently, both operations require the similar amount of time to complete regardless of the size of the target files.

A multimedia file generally consists of a small metadata section and a large frame data section. Because split and merge are generic file system operations and are not aware of the multimedia file structure, when two multimedia files are merged, the metadata sections of the two files must be split first and then unified to make a single metadata section for the final product. The same mechanism applies to the data sections. Naturally, some parts of the metadata sections must be updated to reflect the changes. In this manner, two multimedia files can be combined by a few invocations of the split and merge system calls. Similarly, a multimedia file can be separated easily and efficiently into two independent files by using the proposed operations. These schemes can be used as basic building blocks for more complex multimedia file editing operations, such as trimming or insertion, and require no additional storage space to temporally maintain the partial replication of the original file.

In addition to the performance benefit, the proposed operations will significantly extend the lifespan of the embedded storage system; because they reduce the number of write operations by a significant margin due to the data block copy operations. The proposed operations are implemented for the Ext4, FAT32 and exFAT file systems of the Linux kernel, which are popular file systems for embedded systems and PCs. The virtual file system, which links the user-level application programming interfaces (APIs) and the underlying file systems, is modified to provide transparent support of the proposed operations. Due to this modification, a user-level process can use the proposed operations by simply calling the split and merge systems calls regardless of the underlying file system. This implementation is ported to a commercial smart camera for evaluation.

The remainder of this paper is organized as follows. Section 2 introduces the motivation behind the split/merge operations as well as related work. Section 3 proposes the design principles of the proposed operations. The interface and implementation of the proposed file system are illustrated in Section 4, and the evaluation of the prototype implementation is presented in Section 5. Section 6 concludes the research.

2. Background and related work

2.1. Motivation

Fig. 1 illustrates the conventional trimming scheme for MP4 movie files. The trimmed media file is actually a newly created file, and the frame data of the trimmed media file are replications of the original frame data.

Like many other multimedia codecs, the MP4 specification requires a file to have its type header at the beginning. The type header points to metadata about the files video frames. Usually, the metadata are located at the end of the file. Both header and metadata must reflect the contents of the movie file. Therefore, after creation of the new file, the trimming scheme updates the type header, copies the select frame data, and then appends the updated metadata. The sizes of the type header and metadata are negligible compared to the frame data. Therefore, the majority of the processing time is taken up by the frame copy operation, although the frame data are already stored in the file system, and there is no need to make a redundant copy of them.

To examine the impact of the copy operation, the time to finish the trimming operation was measured. The original file used in this example was 640 MB and 5 min in length. This experiment changed the trimming range from 10 s to 50 s by a 10 s step. In each experiment, the overall response time, number of I/O requests, I/O wait time, and CPU utilization under three different file systems: Ext4, FAT32, and exFAT, were recorded. The details of the experimental configurations are described in Section 5.

As shown in Fig. 2, the response time increased proportional to the selection length, because the longer the selected movie is the more data frame copies occur. Under all three file systems, it took longer than 10 s, which is an unacceptable time for consumer electronics customers to extract a 50 s file. Consequently, the number of I/O requests also increased proportionally to the trimmed file size, as shown in Fig. 3. Even though a significantly long time was required to finish, the actual CPU utilization during the operations was maintained under 15%. This is because most of the CPU time was spent waiting for I/O operations.

The slow response time from the copy-based approach is not just problematic for trimming operations. It applies to multimedia file editing operations in general. For example, injecting a 2 s movie fragment into the beginning of an existing 5 min MP4 file, which is approximately 640 MB, took up to 78 s, according to the experimental results.

2.2. Background

File systems generally group consecutive sectors to form a logical block unit on the storage device [3,4]. This logical block is the smallest data unit that can be allocated for a file.

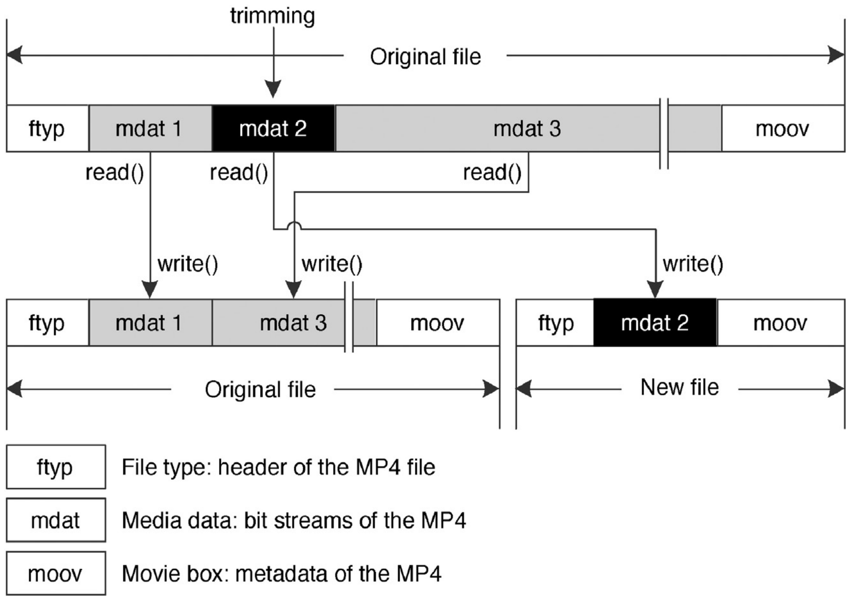

Fig. 1. Conventional copy-based trimming scheme of MP4 files.