Performance Improvement with Zero Copy Technique on FUSE-based Consumer Devices

Junsup Song and Dongkun Shin, *Member, IEEE*
Sungkyunkwan University, Suwon, Korea
song0319@skku.edu, dongkun@skku.edu

Abstract –**FUSE is a framework to develop file systems in user space. Recently, multimedia consumer devices are adopting the FUSE file system to emulate PC-compatible FAT file system. However, FUSE file system has performance overhead due to the data copy between user space and kernel space. In this paper, we improve the performance of FUSE file system by removing the user/kernel data copy with a zero copy technique. Our experiments on a smartphone showed that the zero copy technique improves I/O performance by up to 25%.**

I. INTRODUCTION

Multimedia consumer devices such as MP3 players, digital cameras, and video players provide an external SD card partition in their storage space and support data transfer between the partition and Windows-based PC via USB (Universal Serial Bus) connection. The consumer devices generally use the UMS (USB Mass Storage) protocol in order to allow the connected PC access their data. UMS is a block-level protocol that gives the host PC direct access to the physical blocks on the device storage, and consequently operates below the file system layer. Therefore, the file system of the external SD card partition should be FAT-compatible.

Recent smart devices such as smartphones and tablets have many application codes as well as multimedia data. Device venders generally use a separated internal partition for the application codes in order to prevent host PC from accessing them as shown in Fig. 1(a). In addition, a more optimized file system can be used for the internal partition since FAT file system is not well-optimized in terms of performance and reliability. For example, standard Linux file systems such as EXT4 or NAND flash memory file systems such as YAFFS2 and UBIFS can be used for the internal partition, while the external partition uses a FAT file system to be mounted for UMS. However, the separated partitions present several problems. First, to connect the device with host PC for file transfer, the device should be suspended so that the external partition could be unmounted from the device, and mounted to the PC. Second, the size of each separated partition is fixed. Therefore, there can be an imbalance between the free spaces of two partitions.

A solution for the separated partitions is to use the FUSE (Filesystem in Userspace) [1], which is a pseudo file system that provides a bridge to another file system for selected files. FUSE allows a single partition for both internal data and external data, as only selected files are externally shared via FUSE. The unified partition can be formatted with any file system rather than FAT, as FUSE hides the original file system. Instead of using USB mounts, files are transferred using MTP (Multimedia Transfer Protocol) which operates over the file system layer.

However, the unified partition is not for free. FUSE is composed of a kernel module, called KFM, and a user-level daemon as shown in Fig. 1(b). Since a FUSE file system is a user-level file system, it invokes many context switches and kernel/user mode switches. In addition, the data should be copied several times between kernel memory and user buffer, and there is a double caching problem in the page cache. The context/mode switching and double caching can be mitigated using the FUSE options such as big_writes and direct_io, respectively. The big_writes option enables a large-sized of chunk and thus reduces the number of context/mode switches. The direct_io option removes the kernel buffer space for KFM. However, no solution is provided by FUSE for the data copy overhead between user space and kernel space. In this paper, we adopted the zero copy technique to reduce the data copy overhead. The data in the kernel buffer for EXT4 can be sent directly to the kernel buffer for KFM without passing through the FUSE daemon buffer. Using the splice system call, we implemented the zero copy at FUSE file system.

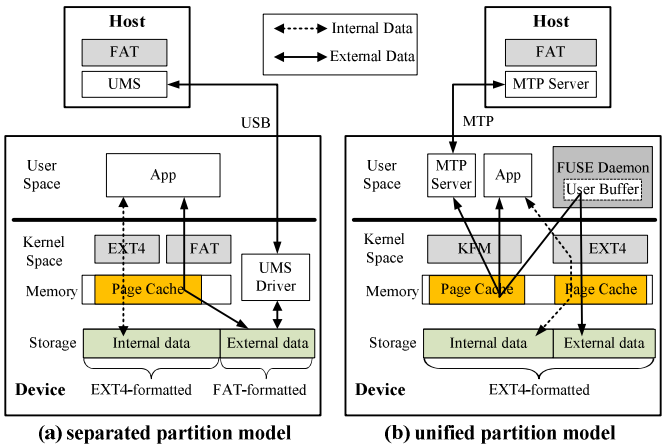

Fig. 1. Two models of externally-mountable partition at multimedia devices

II. ZERO COPY TECHNIQUES

The zero copy technique [2] enables data copy within kernel space without intervention of user application. Three system calls are available in Linux to implement the zero copy, *mmap*, *sendfile*, and *splice*. The *mmap* system call causes the file contents to be copied into a kernel buffer which is shared with

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2012-0006417).

the user process. Then, a following *write* system call causes the kernel to copy the data from the source kernel buffer into the target kernel buffer, without any copy being performed between the kernel and user memory spaces. By using *mmap* instead of *read*, we can reduce the amount of data the kernel has to copy. In addition, it invokes less mode switches compared with a traditional method. However, *mmap* can be used only when the file size is fixed.

With the *sendfile* system call, the file contents can be copied into a kernel buffer, and then the data is copied by the kernel directly into the target kernel buffer. It is an integrated version of *read* and *write* system calls. The *sendfile* system call can utilize the scatter/gather support of DMA (Direct Memory Access) engine. However, it does not support the multiple buffer copy. In FUSE architecture, KFM and FUSE daemon communicate by a FUSE header, which has the request information. Therefore, the FUSE header should be copied between KFM and FUSE daemon, while the data could be copied within kernel by zero copy. To support such an operation, the zero copy technique should support the multiple buffer copy.

The *splice* system call moves data between two file descriptors within kernel address space. Especially, the *vmsplice* system call supports the multiple buffer copy with a vectored IO. For example, the FUSE header from user buffer and the data from kernel buffer can be copied to the target kernel buffer by one *vmsplice* system call as shown in Fig. 2(b).

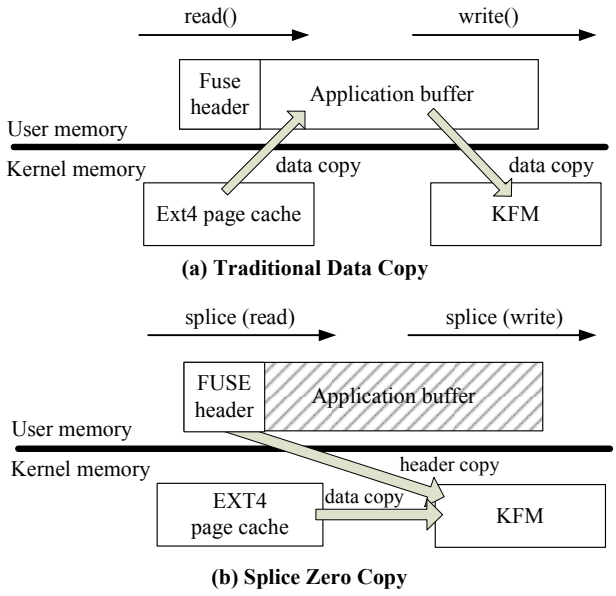

Fig. 2. Comparison between the traditional data copy and the zero copy under FUSE file system.

III. EXPERIMENTS

For our experiments, we used a real smartphone which has a 16GB eMMC flash memory device. It has a unified partition mounted at /data directory, which stores all the application code and data. The partition is mounted via EXT4 file system. There is a subdirectory called /data/sdcard which stores user multimedia files. Only the subdirectory of /data/sdcard can be exposed to MTP via FUSE. We implemented the zero copy with the *vmsplice* system call, and enabled the big_writes option of FUSE. Each experiment was performed after a factory reset in order to initialize the internal state of eMMC storage.

Two benchmark programs were tested. The IOzone is a native benchmark working on shell, and the Quick benchmark is a java application. With the zero copy (ZC) technique, the performances of read and write operations are significantly improved as shown in Fig. 3 and Fig. 4.

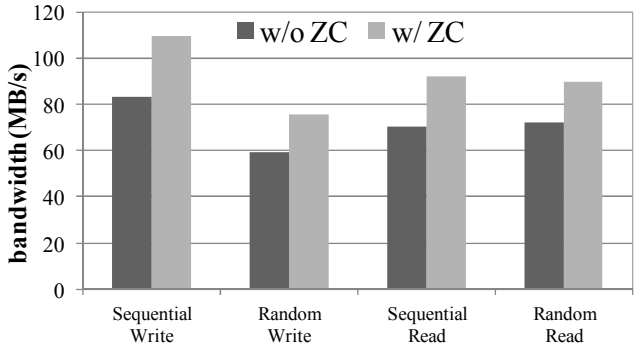

Fig. 3. IOzone benchmark : 20MB file size, 128KB I/O size

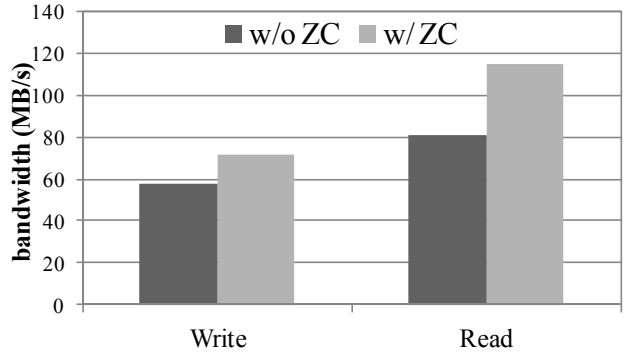

Fig. 4. Quick benchmark : 9MB file size, 1MB I/O size

IV. CONCLUSION

While FUSE enables a unified file system to the multimedia devices which should share only a part of their data with a connected PC, FUSE has several performance problems. In this paper, we applied the zero copy technique for the FUSE-based multimedia devices to reduce the data copy between the kernel and user buffers. Our experiments show the zero copy can improve performance by up to 25% on benchmark programs.

REFERENCE

[1] Aditya Rajgarhia and Ashish Gehani, "Performance and extension of user space file systems," Proc. of the 2010 ACM Symposium on Applied Computing, pp. 206-213, 2010.

[2] Piyush Shivam, Pete Wyckoff, and Dhabaleswar Panda, "EMP: Zero-Copy OS-Bypass NIC-Driven Gigabit Ethernet Message Passing," Proc. of the 2001 ACM/IEEE conference on Supercomputing, 2012.