To FUSE or Not To FUSE: Performance of User-Space File Systems

Bharath Kumar Reddy Vangoor, Vasily Tarasov, Erez Zadok

In 2017 USENIX Conference on File and storage Technologies (FAST '17)

2021. 03. 17

Presentation by Han, Yejin

hyj0225@dankook.ac.kr

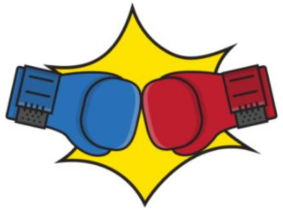
Contents

Controversy on Userspace file system


```
Umm.

"userspace filesystem"?
                                   ...
fuse, and thus _should_ be done with fuse is just ridiculous. That's
like saying you should do a microkernel - it may sound nice on paper,
but it's a damn stupid idea for people who care more about some idea
than they care about reality.

                          Linus
```



User-space file systems

- Development Ease
- Portability
- Libraries
- Existing Code and User Base

?

performance overhead

FUSE high-level architecture

- Linux kernel module(fuse.ko), FUSE driver, User-space daemon

*Filesystem daemon: the process providing the data and metadata of the filesystem

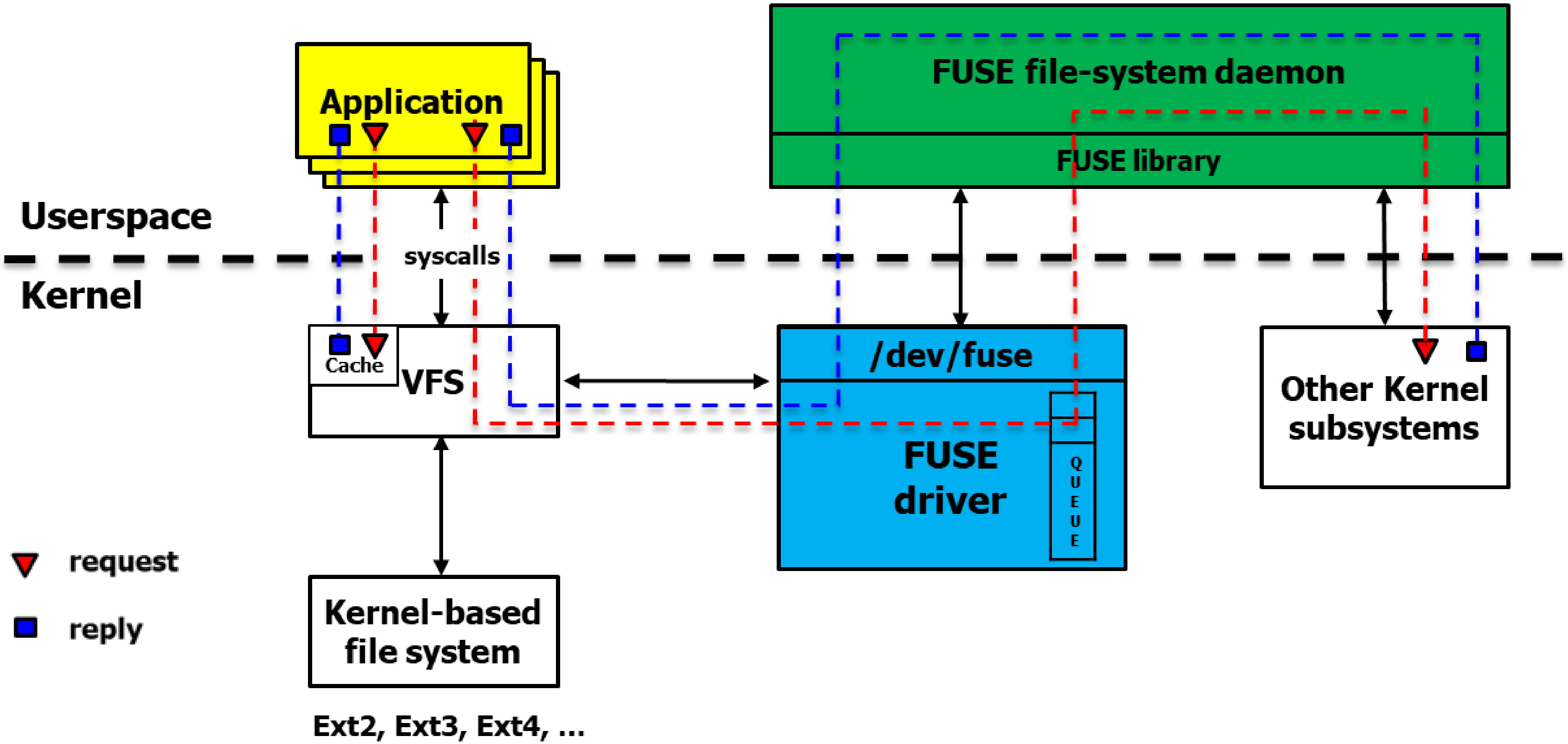
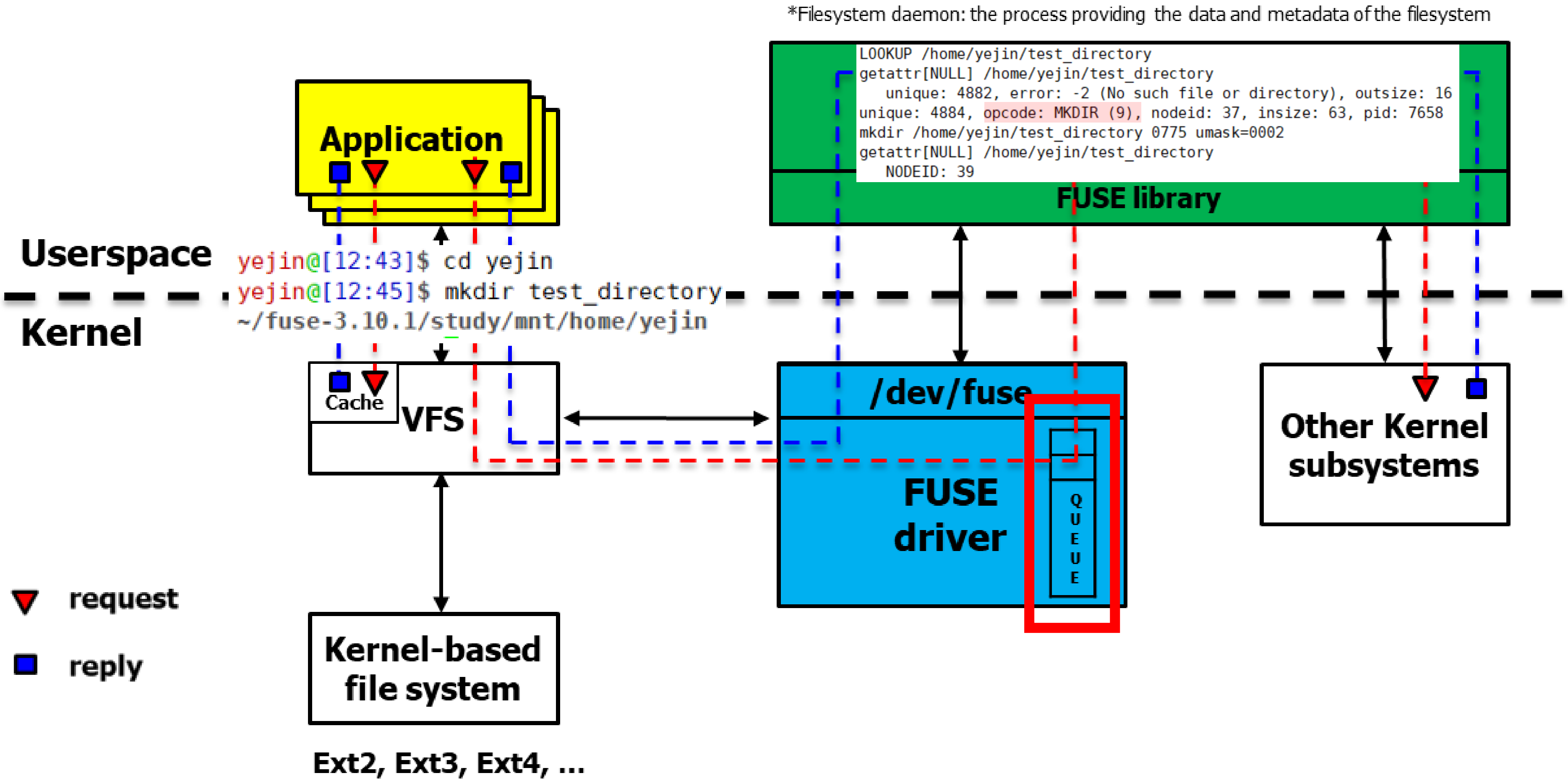


Figure 1: FUSE high-level architecture.

FUSE high-level architecture

- Linux kernel module(fuse.ko), FUSE driver, User-space daemon



FUSE Request

- User-kernel protocol, Request types

Linux Kernel: include/uapi/linux/fuse.h
User level lib: include/fuse_kernel.h

```
  * 7.23
  *  - add FUSE_WRITEBACK_CACHE
               ...
enum fuse_opcode {
    FUSE_LOOKUP      = 1,
    FUSE_FORGET      = 2,  /* no reply */
    FUSE_GETATTR         = 3,
    FUSE_SETATTR         = 4,
    FUSE_READLINK        = 5,
    FUSE_SYMLINK         = 6,
    FUSE_MKNOD       = 8,
    FUSE_MKDIR       = 9,
    FUSE_UNLINK      = 10,

         ⋮
```

Group (#)	Request Types
Special (3)	**INIT**, **DESTROY**, **INTERRUPT**
Metadata (14)	**LOOKUP**, **FORGET**, **BATCH_FORGET**, CREATE, UNLINK, LINK, RENAME, RENAME2, **OPEN**, **RELEASE**, STATFS, FSYNC, **FLUSH**, **ACCESS**
Data (2)	READ, WRITE
Attributes (2)	GETATTR, SETATTR
Extended Attributes (4)	SETXATTR, GETXATTR, LISTXATTR, REMOVEXATTR
Symlinks (2)	SYMLINK, READLINK
Directory (7)	MKDIR, RMDIR, **OPENDIR**, **RELEASEDIR**, READDIR, **READDIRPLUS**, FSYNCDIR
Locking (3)	GETLK, SETLK, SETLKW
Misc (6)	BMAP, FALLOCATE, MKNOD, IOCTL, POLL, NOTIFY_REPLY

FUSE API levels

- High-level vs. Low level API

High-level API

```
struct fuse_operations {
        int (*getattr) (const char *, struct stat *, struct fuse_file_info *fi);

        int (*readlink) (const char *, char *, size_t);

        int (*mknod) (const char *, mode_t, dev_t);

        int (*mkdir) (const char *, mode_t);

        int (*unlink) (const char *);

        int (*rmdir) (const char *);
```

⋮

- Skip the path-to inode mapping
- work with file names and paths instead of inodes
- Development ease

Low-level API

```
struct fuse_lowlevel_ops {
        void (*init) (void *userdata, struct fuse_conn_info *conn);

        void (*destroy) (void *userdata);

        void (*lookup) (fuse_req_t req, fuse_ino_t parent, const char *name);

        void (*forget) (fuse_req_t req, fuse_ino_t ino, uint64_t nlookup);

        void (*getattr) (fuse_req_t req, fuse_ino_t ino,
                         struct fuse_file_info *fi);
```

⋮

- Receiving and parsing requests from the kernel
- work with inodes
- High flexibility

FUSE Queues

- Interrups, forgets, pending, processing, background

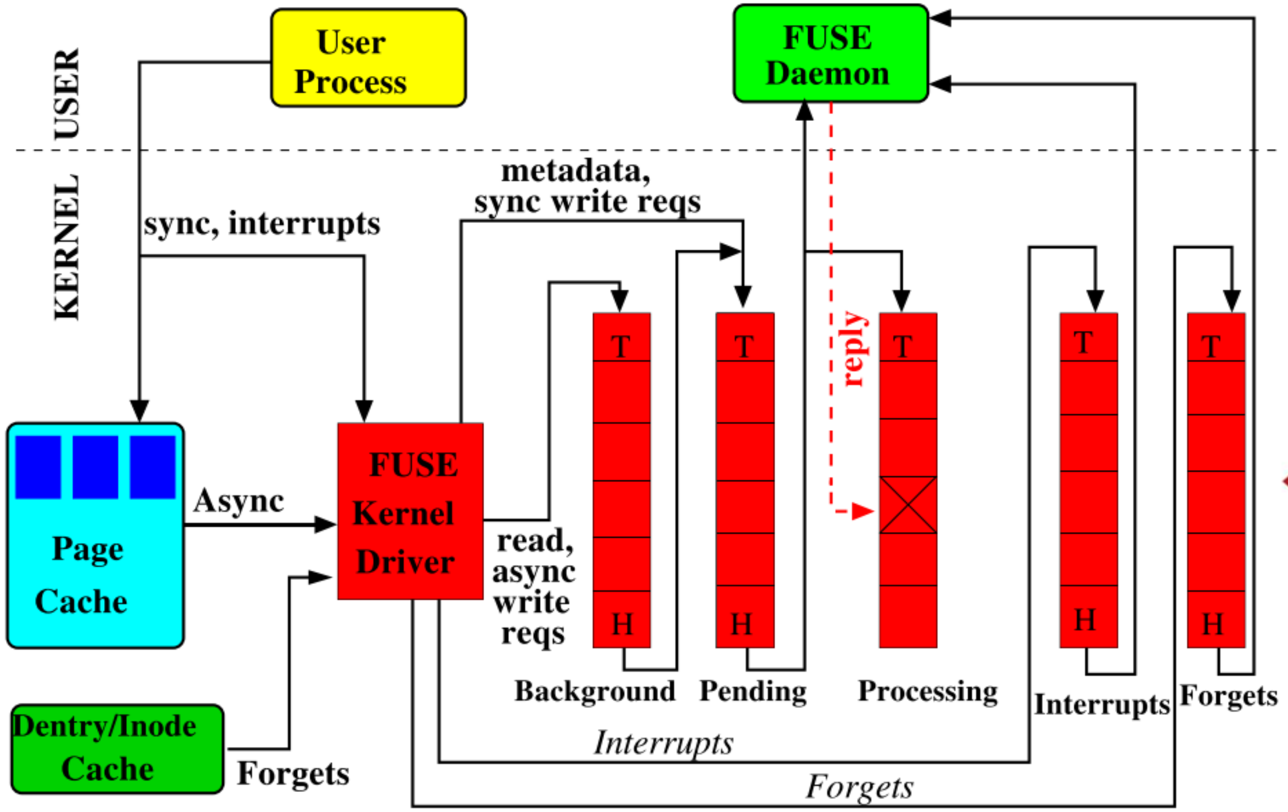


Figure 2: The organization of FUSE queues marked with their Head and Tail. The processing queue does not have a tail because the daemon replies in an arbitrary order.

Request Types
INIT, DESTROY, INTERRUPT
LOOKUP, FORGET, BATCH_FORGET, CREATE, UNLINK, LINK, RENAME, RENAME2, **OPEN, RELEASE,** STATFS, FSYNC, **FLUSH, ACCESS**
READ, WRITE
GETATTR, SETATTR
SETXATTR, GETXATTR, LISTXATTR, REMOVEXATTR
SYMLINK, READLINK
MKDIR, RMDIR, **OPENDIR, RELEASEDIR,** READDIR, **READDIRPLUS,** FSYNCDIR
GETLK, SETLK, SETLKW
BMAP, FALLOCATE, MKNOD, IOCTL, POLL, NOTIFY_REPLY

FUSE Optimizations

Zero copy using Splicing

- /dev/fuse read()/write() requires a memory copy between the kernel and user space.
- Since they often process a lot of data, FUSE use **splicing** functionality.
- transfer data without copying the data

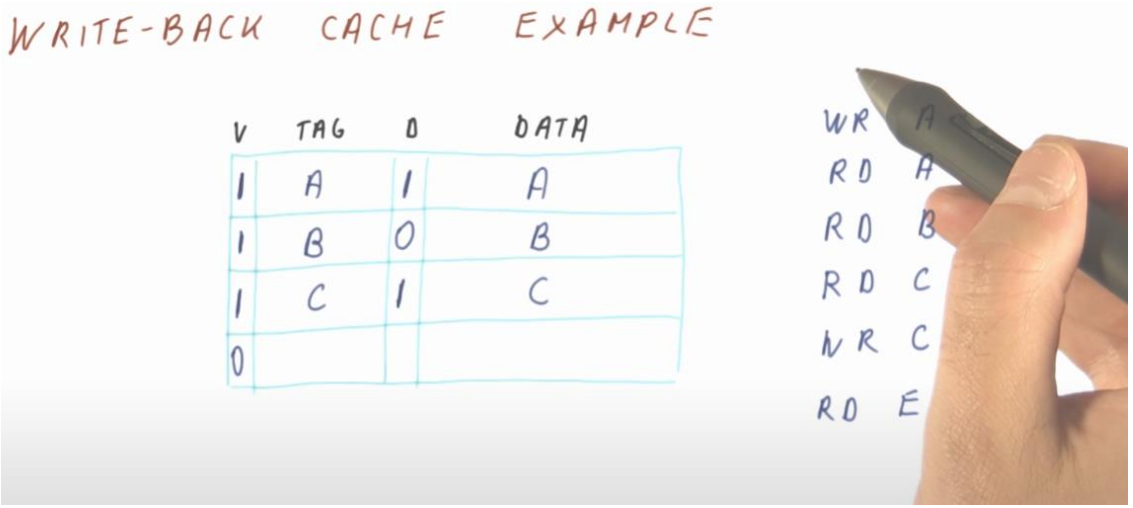
Multi-threaded daemon

- If there are more requests in the pending queue, FUSE spawns additional threads.
- Every thread processes one request at a time.
- After processing the request, each thread checks if there are more than 10 threads running;
  If so, that thread exits.

FUSE Optimizations

Write-back cache

- The basic write behavior of FUSE is synchronous and only 4KB of data is sent to the user daemon.
- When copying a large file, it can result in performance problems.
- So FUSE support a write-back policy and make writes asynchronous.
- File data can be pushed to the user daemon in larger chunks.



StackFS

- Stackable passthrough file system using FUSE's low-level API
- StackFS layers on top of an Ext4 file systems
- StackFS passes FUSE requests directly to the Ext4

https://github.com/sbu-fsl/fuse-stackfs

Evaluation Environment

CPU	Intel Xeon CPU E5530 4 core 2.40GHz processor
Memory	4 GB RAM
OS	64-bit Linux 3.14
Filesystem	Ext4 / *StackfsBase / *StackfsOpt
Device	200GB Intel X25-M SSD/ 146GB Seagate Savvio 15K.2 15KRPM HDD
Workload	Sequential/random-read/write-N threads-M files

*StackfsBase: with no major FUSE optimizations
*StackfsOpt: with all FUSE improvements
(write-back cache, increased size of request, multi-threaded, splicing)

Performance Observation

- The Relative difference varied across workloads, devices, and FUSE configurations
- FUSE’s optimizations improve performance significantly

#	Workload	I/O Size (KB)	HDD Results			SSD Results		
			EXT4 (ops/sec)	StackfsBase (%Diff)	StackfsOpt (%Diff)	EXT4 (ops/sec)	StackfsBase (%Diff)	StackfsOpt (%Diff)
37	files-cr-1th	4	30211	- 57!	- 81.0!	35361	- 62.2!	- 83.3!
45	web-server	-	1704	- 51.8!	+6.2+	19437	- 72.9!	-17.3*

Performance Observation

- But optimizations can degrade the performance on other workloads.
- Only two file-create workloads fell into the red group (50%↑ degration)

#	Workload	I/O Size (KB)	HDD Results			SSD Results		
			EXT4 (ops/sec)	StackfsBase (%Diff)	StackfsOpt (%Diff)	EXT4 (ops/sec)	StackfsBase (%Diff)	StackfsOpt (%Diff)
39	files-rd-1th	4	645	+ 0.0+	- 10.6*	8055	- 25.0*	- 60.3!
37	files-cr-1th	4	30211	- 57!	- 81.0!	35361	- 62.2!	- 83.3!
38	files-cr-32th	4	36590	- 50.2!	- 54.9!	46688	- 57.6!	- 62.6!

Performance Observation

- Performance depends significantly on the underlying device.
- Stackfs performs visibly worse for metadata-intensive and macro workloads (especially low for SSDs)

#	Workload	I/O Size (KB)	HDD Results			SSD Results		
			EXT4 (ops/sec)	StackfsBase (%Diff)	StackfsOpt (%Diff)	EXT4 (ops/sec)	StackfsBase (%Diff)	StackfsOpt (%Diff)
37	files-cr-1th	4	30211	- 57!	- 81.0!	35361	- 62.2!	- 83.3!
38	files-cr-32th	4	36590	- 50.2!	- 54.9!	46688	- 57.6!	- 62.6!
39	files-rd-1th	4	645	+ 0.0+	- 10.6*	8055	- 25.0*	- 60.3!
40	files-rd-32th	4	1263	- 50.5!	-4.5+	25341	- 74.1!	-33.0#
41	files-del-1th	-	1105	- 4.0+	- 10.2*	7391	- 31.6#	- 60.7!
42	files-del-32th	-	1109	- 2.8+	- 6.9*	8563	- 42.9#	- 52.6!
43	file-server	-	1705	- 26.3#	-1.4+	5201	- 41.2#	-1.5+
44	mail-server	-	1547	- 45.0#	-4.6+	11806	- 70.5!	-32.5#
45	web-server	-	1704	- 51.8!	+6.2+	19437	- 72.9!	-17.3*

workloads

- Read workloads
- Write workloads

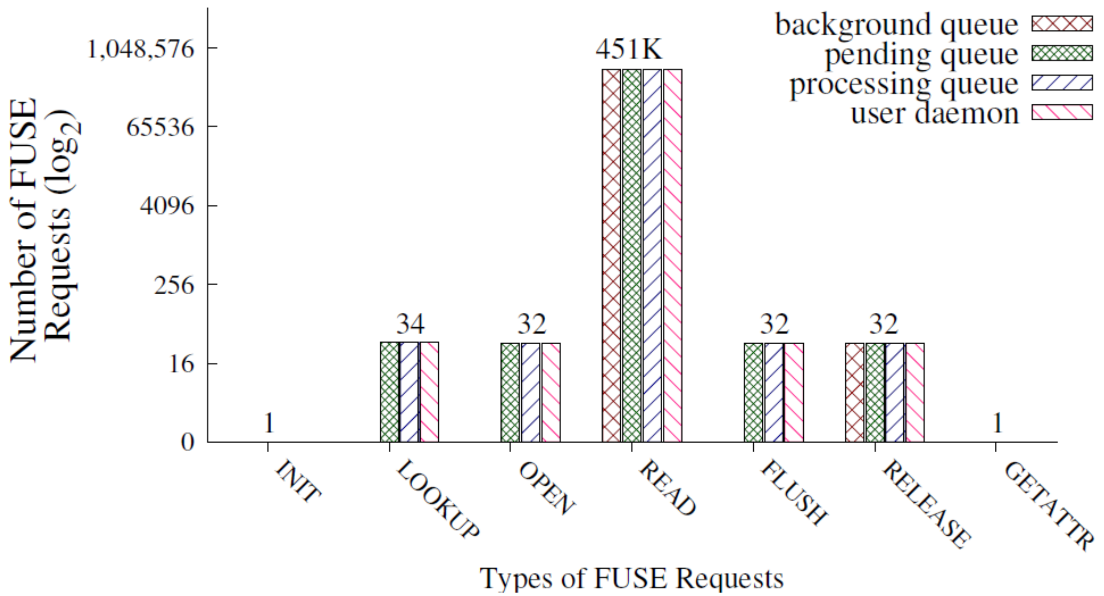


Figure 3: Different types and number of requests generated by StackfsBase on SSD during the `seq-rd-32th-32f` *work-load, from left to right, in their order of generation.*

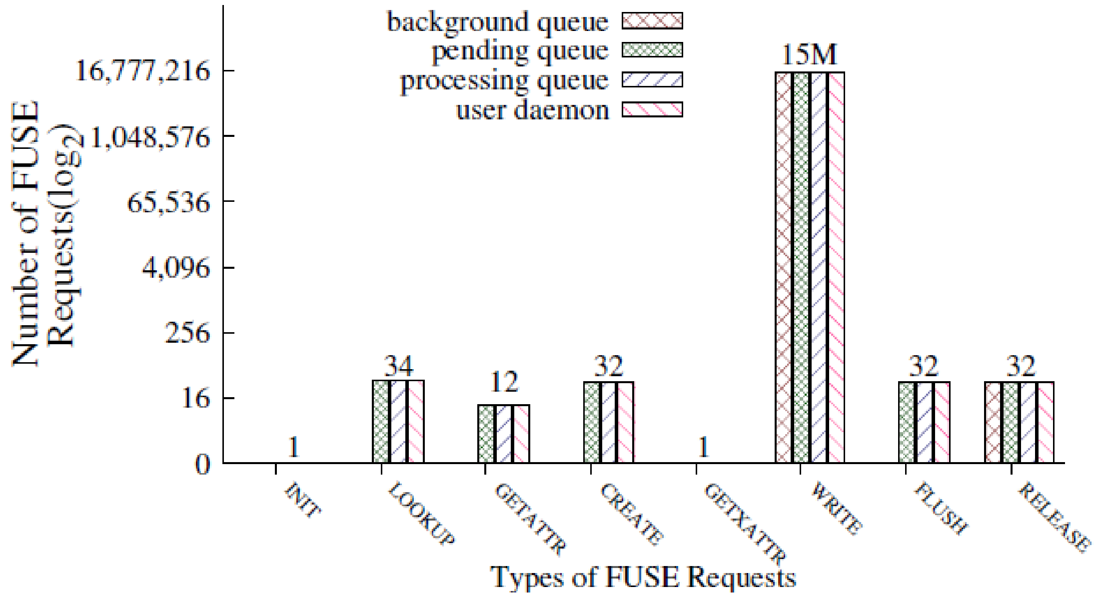
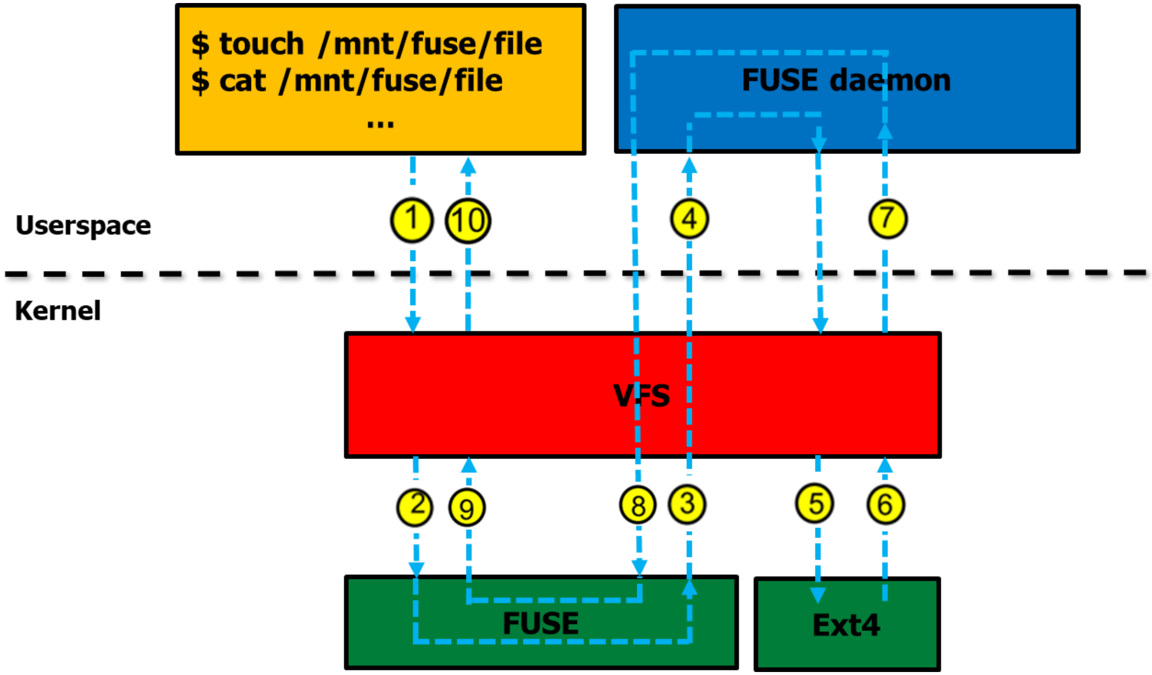


Figure 4: Different types of requests that were generated by StackfsBase on SSD for the `seq-wr-32th-32f` *workload, from left to right in their order of generation.*

- Performance tracing points



Stages of `write()` call processing	Time (μs)	Time (%)
Processing by VFS before passing execution to FUSE kernel code	1.4	2.4
FUSE request allocation and initialization	3.4	6.0
Waiting in queues and copying to user space	10.7	18.9
Processing by Stackfs daemon, includes Ext4 execution	24.6	43.4
Processing reply by FUSE kernel code	13.3	23.5
Processing by VFS after FUSE kernel code	3.3	5.8
Total	56.7	100.0

Table 4: Average latencies of a single write request generated by StackfsBase during seq-wr-4KB-1th-1f workload across multiple profile points on HDD.

- Metadata workloads

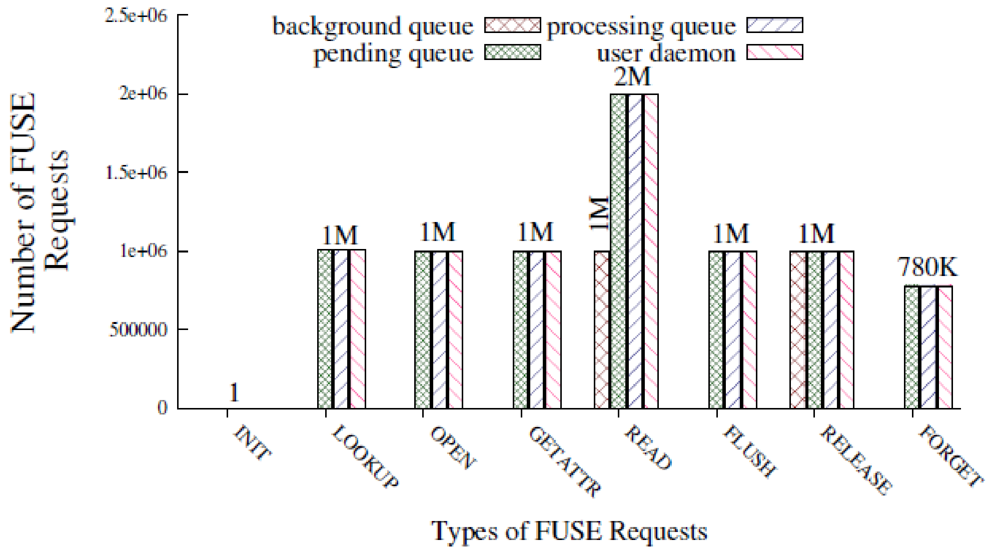


Figure 5: Different types of requests that were generated by StackfsBase on SSD for the `files-rd-1th` workload, from left to right in their order of generation.

- Macro server workloads

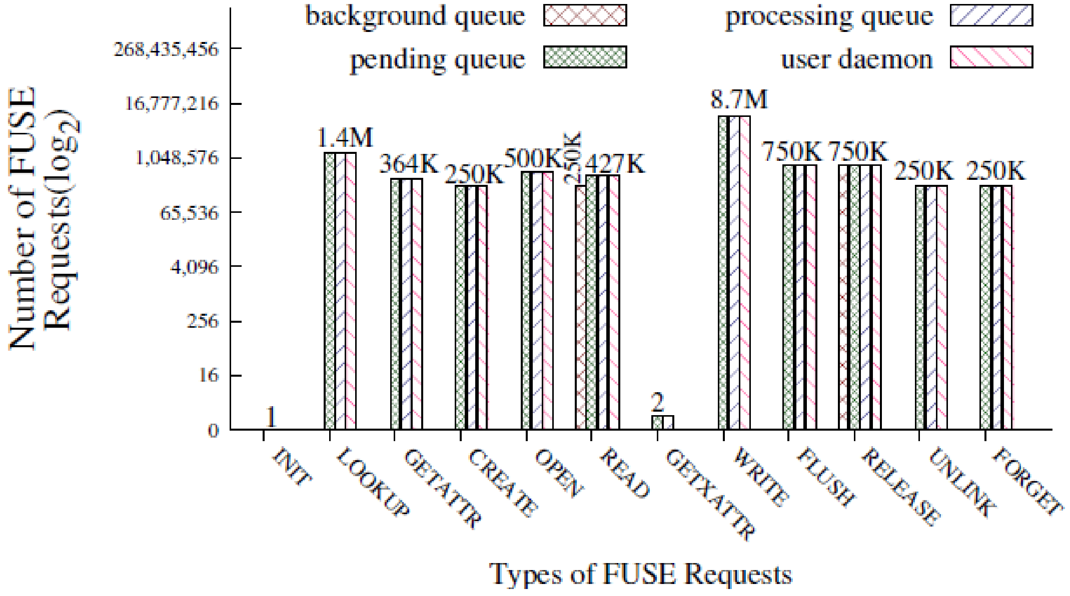


Figure 6: Different types of requests that were generated by StackfsBase on SSD for the `file-server` workload.

- FUSE is one of the most popular user-space file system frameworks
- In-depth performance analysis of FUSE performance
- Depending on the workload/hardware, FUSE performance degration can be imperceptible or as high as -83%

To FUSE or Not To FUSE: Performance of User-Space File Systems

Bharath Kumar Reddy Vangoor, Vasily Tarasov, Erez Zadok

In 2017 USENIX Conference on File and storage Technologies (FAST '17)

Thank You!

2021. 03. 17
Presentation by Han, Yejin
hyj0225@dankook.ac.kr