

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

kSFS: Repurposing a Microkernel-like Interface for Fast and Secure In-Kernel Linux File Systems

Dinglan Peng and Pedro Fonseca, *Purdue University*

https://www.usenix.org/conference/usenixsecurity26/presentation/peng-dinglan

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium is sponsored by



جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology



USENIX Security '26 Artifact Appendix: kSFS: Repurposing a Microkernel-like Interface for Fast and Secure In-Kernel Linux File Systems

Dinglan Peng
Purdue University

Pedro Fonseca
Purdue University

A Artifact Appendix

A.1 Abstract

This artifact contains the code and scripts for building kSFS and its dependencies, and for comparing it with other systems.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Running the experiments with this artifact may cause data loss. We suggest using a new machine without important data.

A.2.2 How to access

The artifacts of kSFS can be accessed at https://github.com/rssys/ksfs-artifacts/tree/9aa1c814524ea146df78e2023b9221bd6755574d or https://doi.org/10.5281/zenodo.17973522.

A.2.3 Hardware dependencies

To evaluate this artifact, a server with at least a 24-core CPU, 128 GiB RAM, and an NVMe SSD is required. If the artifact is run on a VM, the `FSGSBASE` CPU feature should be enabled, which can be done by setting CPU model to `host` on QEMU.

A.2.4 Software dependencies

Running the experiments requires a fresh installation of Debian 12 (or newer versions) or Ubuntu 24.04 (or newer versions).

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

Download to the artifact and run `install.sh` inside it to install all the dependencies including the kSFS kernel. Alternatively, you can manually build the kernel and install required software following the script. After installation, reboot the server with the installed kernel, and then run `build.sh` to build file systems for kSFS and other comparison systems. Please ignore the errors produced by installing libfuse inside the `fuse-opt` directory. For generating figures, the following Python packages are required: `pandas` and `matplotlib`, which can be installed with `pip`. Edit `experiments/env.sh` to configure the disk and the mount point to use in the experiments. The specified disk will be wiped.

A.3.2 Basic Test

Run the following command to create a file system on a disk and mount it with kSFS: `sudo ./drivers/mount.sh ksfs ntfs <disk> <mount point> --mkfs`. Please specify a disk and a mount point for testing. The disk will be wiped.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): kSFS achieves performance better than or in the same order of magnitude as other systems, including FUSE and Bento, for the work loads described in Section 7 whose results are illustrated in Figure 3, 4, 5, 6 and Table 4, depending on the underlying hardware.

(C2): kSFS can catch the injected faults described in Section 7.5 whose results are illustrated in Table 6.

A.4.2 Experiments

Before running experiments, please remove the existing raw data in `experiments/results` to avoid conflicts.

(E1): Sequential Read/Write Performance [30 human-minutes + 2 compute-hour]: compare the performance of kSFS with other systems on sequential read/write performance for C1.

Execution: Run `run.sh`, `run_bento.sh`, and `run_ext4.sh` in `experiments/fio-sequential`.

Results: Run `fio-sequential.py` in `experiments/figures` to generate Table 4.

(E2): Random Read/Write Performance [30 human-minutes + 10 compute-hour]: compare the performance of kSFS with other systems on random read/write performance for C1.
Execution: Run `run.sh`, `run_bento.sh`, and `run_ext4.sh` in `experiments/fio-rand`.
Results: Run `fio-rand.py` in `experiments/figures` to generate `fig-fio-rand.pdf` for Figure 3.

(E3): Filebench Performance [30 human-minutes + 6 compute-hour]: compare the performance of kSFS with other systems on the filebench benchmark tool for C1.
Execution: Run `run.sh`, `run_bento.sh`, and `run_ext4.sh` in `experiments/filebench`.
Results: Run `filebench.py` in `experiments/figures` to generate `fig-filebench.pdf` for Figure 4.

(E4): Small File Performance [30 human-minutes + 6 compute-hour]: compare the performance of kSFS with other systems on small file operations for C1.
Execution: Run `run.sh`, `run_bento.sh`, and `run_ext4.sh` in `experiments/tar`.
Results: Run `tar.py` in `experiments/figures` to generate `fig-tar.pdf` for Figure 5.

(E5): RocksDB Performance [30 human-minutes + 20 compute-hour]: compare the performance of kSFS with other systems on RocksDB for C1.
Execution: Run `run.sh`, `run_bento.sh`, and `run_ext4.sh` in `experiments/rocksdb`.
Results: Run `rocksdb.py` in `experiments/figures` to generate `fig-rocksdb.pdf` for Figure 6.

(E6): Fault Injection [30 human-minutes + 30 compute-minutes]: test kSFS's fault handling for C2.
Execution: Run `run.sh` in `experiments/fault-injection`.
Results: All outputs in `experiments/results/fault-injection` should contain `fault caught`.

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2026/.