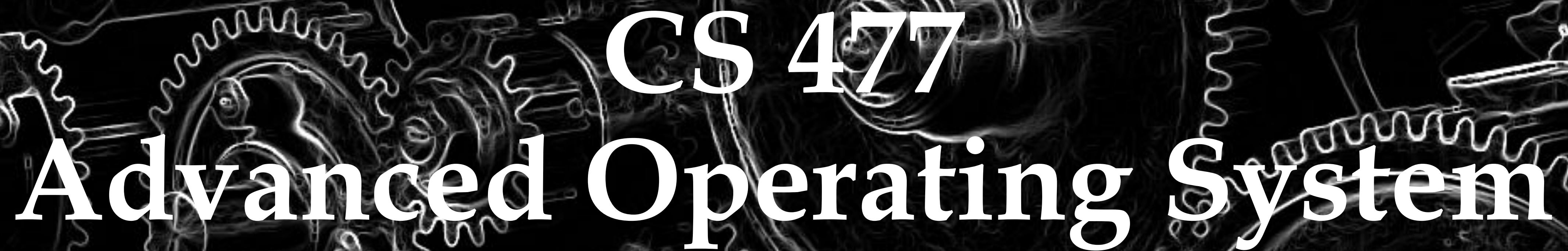


Tutorial 11: File System History and APIs

EPFL

Today's tutorial

- Why shall I learn this?
- "Everything is a file" philosophy
- File System APIs
- Examples of file systems
  - Devfs
  - Sysfs
  - Procfs
  - Initramfs
- All the demo code is here:https://github.com/PanJason/fs_demo

EPFL

Why shall I learn these things, like file system?

Well, compared to mm, if you work as an engineer, you will have more chances of interacting with the file system even if you don't need to store anything…

Some examples if you work directly on infra:

- You work on logging system
- You work on DB

Some examples if you work on higher layers:

- Your application needs to persist user data
- Your application needs to run an LLM

Some examples unrelated to storage:

- You need to expose some interface for your kernel functionality
- You need to enable a new peripheral on Linux

EPFL

Why file abstraction?

Let’s date back to 1950 ~ 1960s. Different APIs for different devices

- Disk: “channel programs”; Tapes: read block, rewind…

Nothing is portable:

- If I wrote a program for this disk, it most likely will not work for another disk

Why file abstraction?

Record format on different devices:

- Tape: sequential records
- Disk: fixed-size blocks
- TTY: char stream
- Card reader: 80-byte blocks

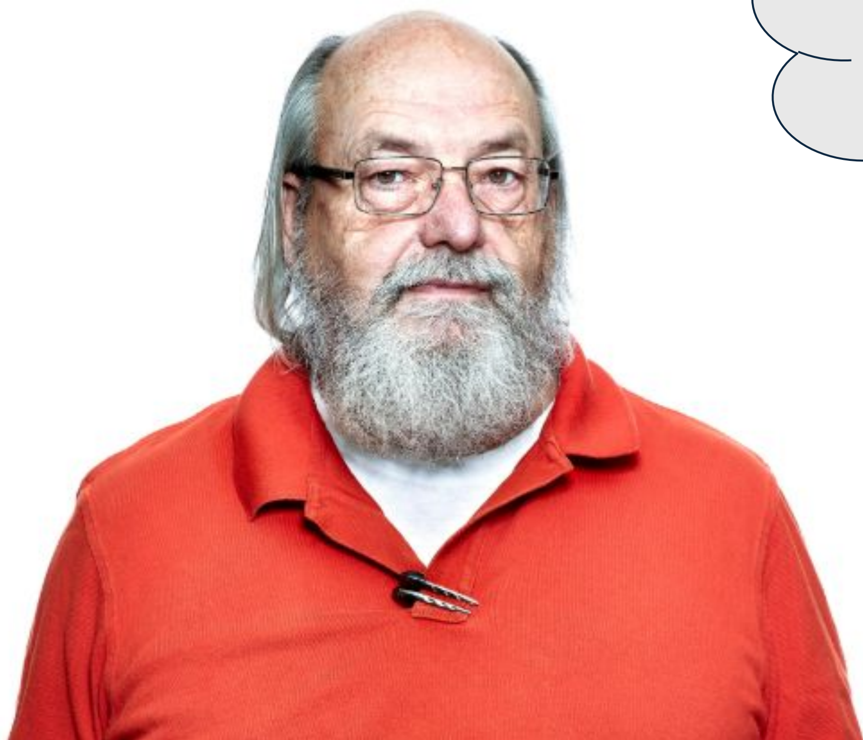
So a program, even the OS, needs to be ported to all possible devices

=> Strong software/hardware coupling!

EPFL

Why file abstraction?

This is so much pain in programming!



In early Unix (PDP-7):
Let's represent any stream of bytes as a file, no matter what it really is.

Now:
- Same semantics for disks, serial ports, pipes, sockets, kernel interfaces (/proc, /sys)
- Drivers become pluggable into a VFS layer
- Portability, simplicity, elegance
- IOCTL resolves the non portable part

EPFL

File Interface

So here comes the file abstractions:

Syscall interfaces (fd):

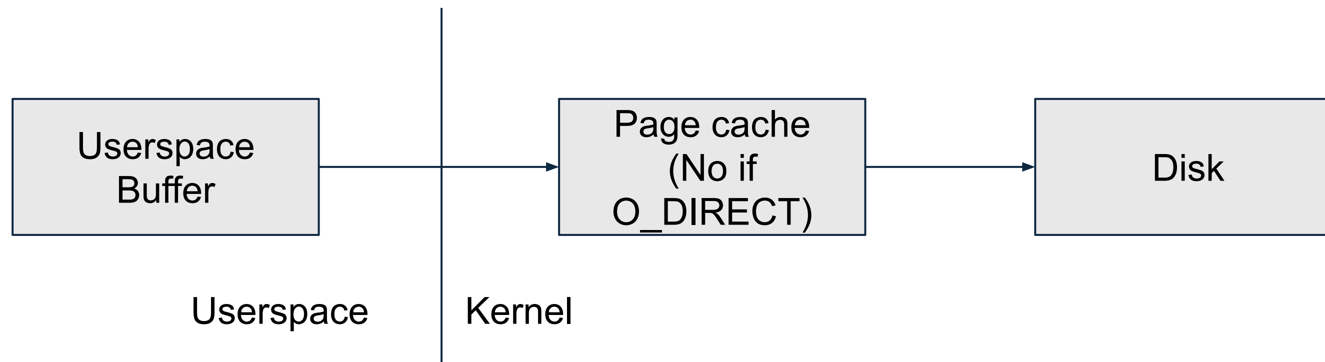
- open, read, write, pread, pwrite, close, lseek

Pros:

- Simple and portable (POSIX standard)
- Works for files, pipes, sockets, devices, procfs, sysfs, epoll, select …
    - Basically everything with fd

Cons:

- Small I/O -> frequent syscall -> not good for high IOPS



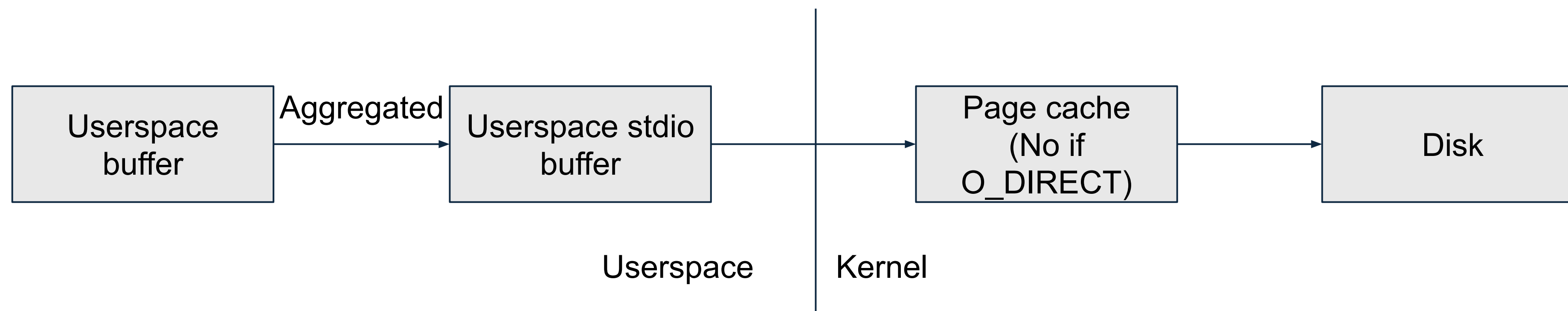
EPFL

File Interface

C Standard I/O Library (FILE*)
- fopen, fread, fwrite, fflush, fseek, fclose
- A wrapper over POSIX ones of the previous slide

Pros:
- Use stdio buffer (userspace) so faster for small I/Os
- Support formatted I/O (printf/scanf)

Cons:
- Another buffer
- File specific



EPFL

File Interface

System call (mem):

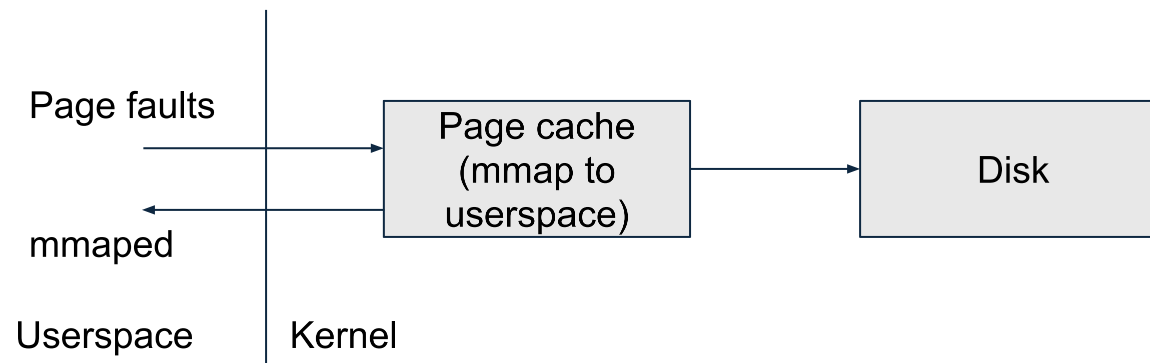
- Mmap, munmap, msync (with page faults)

Pros:

- No user-kernel copy (good for random I/O)
- No syscalls

Cons:

- Page fault cost (fault, PTE creation, TLB shootdown in clear)
- Not good for streaming (minor fault is worse, well there are also mitigations like do_fault_around, which installs 16 pages ahead)



EPFL

File Interface

io_uring:
- Submission queue and completion queues
- No syscalls after set up

Pros:
- Faster async I/O performance

Cons:
- More complex and requires recent kernels
- The complexity in both programming, performance tuning and error handling

EPFL

Comparison

APIs:	Good for	Bad for
read/write	sequential , buffered, predictable, large I/O	Random, small I/O
fread/fwrite	Sequential, buffered, predictable, small I/O	Random, large I/O
mmap	random access, small working sets, DB index reads, caching hot data	Sequential large I/O
io_uring	Async streaming I/O	Cases which don’t want complexity

Some specific file system examples

Some examples of linux virtual file systems that are used to expose kernel states and device information

DevFS:

A special filesystem mounted at /dev

- Kernel automatically created device nodes when hardware appeared
- Kernel creates inodes for devices

Some examples:

- /dev/sda
- /dev/null
- /dev/tty
- /dev/kvm
- /dev/nvidia0

EPFL

Some specific file system examples

Sysfs:

- A file system mounted at /sys that exposes the kernel object (kobject) hierarchy

How does it work:

- Each device/driver/bus/class in the kernel is represented as a kobject
- Whenever a kobject is registered, sysfs creates a directory under /sys.
- Each kobject has a bunch kobj_attribute
- kobject_add (sysfs_create_dir), sysfs_create_file

Some examples:

- /sys/devices/
- /sys/class/net/
- /sys/block/
- /sys/kernel/mm/lru_gen/
- /sys/fs/cgroup/

/sys/kernel/debug/tracing/trace is debugfs actually

EPFL

Some specific file system examples

ProcFS:

- Exposes dynamic runtime kernel state such as memory info, processes, mounts
- proc_create with proc_ops

Some examples:

- /proc/meminfo
- /proc/cpuinfo
- /proc/interrupts
- /proc/<pid>/stack

EPFL

Some specific file system examples

InitramFS

- The purpose of having initramFS is to break the circular dependency:
    - Need some third party to load the disk driver to load the root filesystem
- A compressed cpio archive, loaded by bootloader to perform early userspace initialization
    - Loading the kernel module required to mount a minimal root filesystem
    - Decrypting disk

Workflow:

1. Bootloader loads kernel + initramfs image
    a. Vmlinuz + initramfs.cpio.gz
2. Kernel creates an empty rootfs
3. Kernel unpacks initramfs into rootfs
4. Executes /init from initramfs
5. Load kernel module and mount the real rootfs
6. Switch to newroot.