Storage

```
Course Code: ELEE1119

Course Name: Advanced Computer Engineering

Credits: 30

Module Leader: Seb Blair BEng(H) PGCAP MIET MIHEEM FHEA
```

Storage Types

SSD: don’t rely on magnets and disks, instead they use a type of flash memory called NAND. In an SSD, semiconductors store information by changing the electrical current of circuits contained within the drive. This means that unlike HDDs, SSDs don’t require moving parts to operate.

HDD: Use magnetic technology to store data and has been around since the 1950. A hard disk drive is comprised of a stack of spinning metal disks known as platters. Each spinning disk has trillions of tiny fragments that can be magnetised in order to represent bits (1s and 0s in binary code).

Storage Types

Optical Storage: CD can store up to 700 MB of data, DVD-DL can store up to 8.5 GB, and Blu-Ray can store between 25 and 128 GB of data.

Flash Memory: A flash memory device contains trillions of interconnected flash memory cells that store data. These cells hold millions of transistors that when switched on or off represent 1s and 0s in binary code, allowing a computer to read and write information. Currently can store 2 TB of data.

Types of SSD

- Serial Advanced Technology Attachment (SATA)
- Mini SATA
- SATA M.2

Types of SSD

- Periphal Component Interconnect Express M.2 (PCIE)
- Nov-Volatile Memory Express M.2 (NVME)
-

Comparison

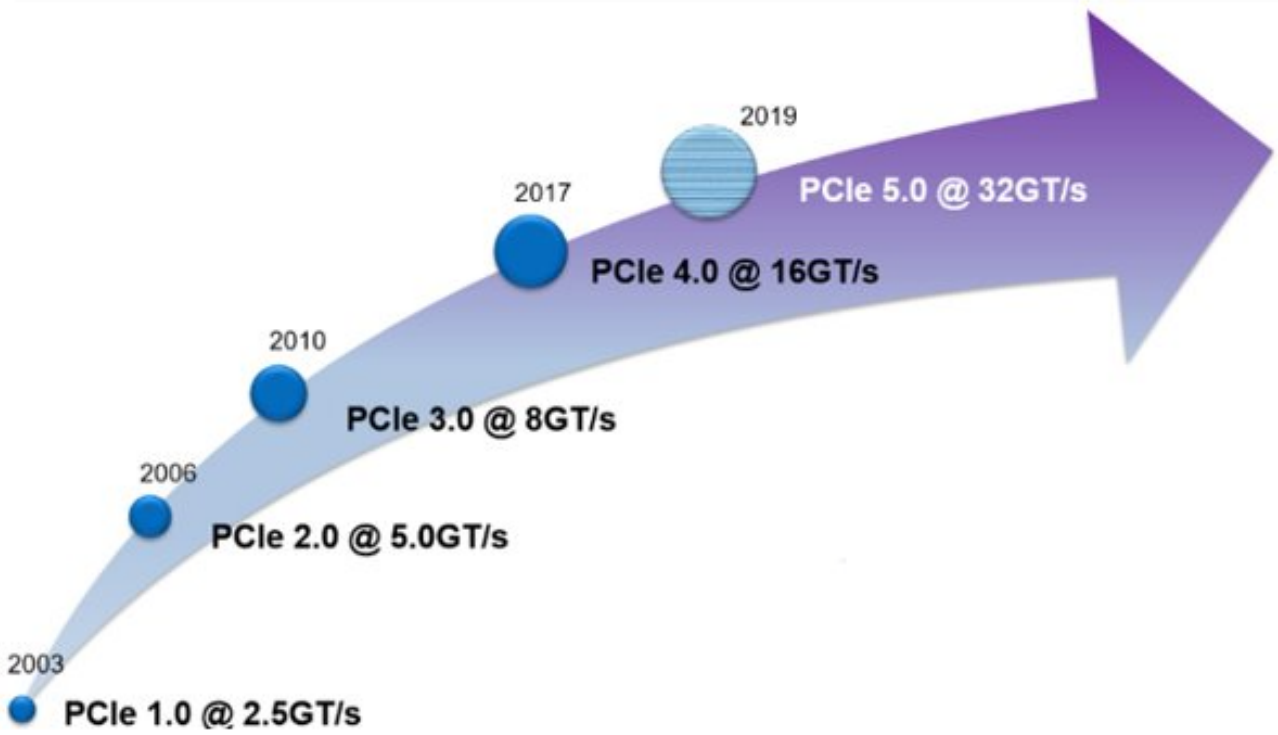
	NVME SSD	M.2 SSD	SATA SSD
Speed	PCIE Gen 3 < = 3.5GB/s PCIE Gen 4 <= 7.5GB/s	<= 550MB/s	<= 550MB/s
Form Factor	M.2, U.2, PCIE	M.2	2.5" M.2
Interface Types	NVME	SATA, NVME	SATA
Adv	High Speeds	Little physical space	Good balance between affordability and speed
Dis-Adv	High Costs	More expensive 2.5"	Slower speeds

PCIE Gen

- 2.0 - 8b/10b encoding.
- 3.0 - 128b
- 4.0 - 130b encoding technique

4.0's 2 extra bits allows for reasonable clock recovery (extracting timing information from the datastream) and ensures alignment of the datastream. 16GT/s (gigatransfers) which means the maximum theoretical bandwith of PCIe 4.0 is:

15.754Gb^{-s} and 1.54% overhead.

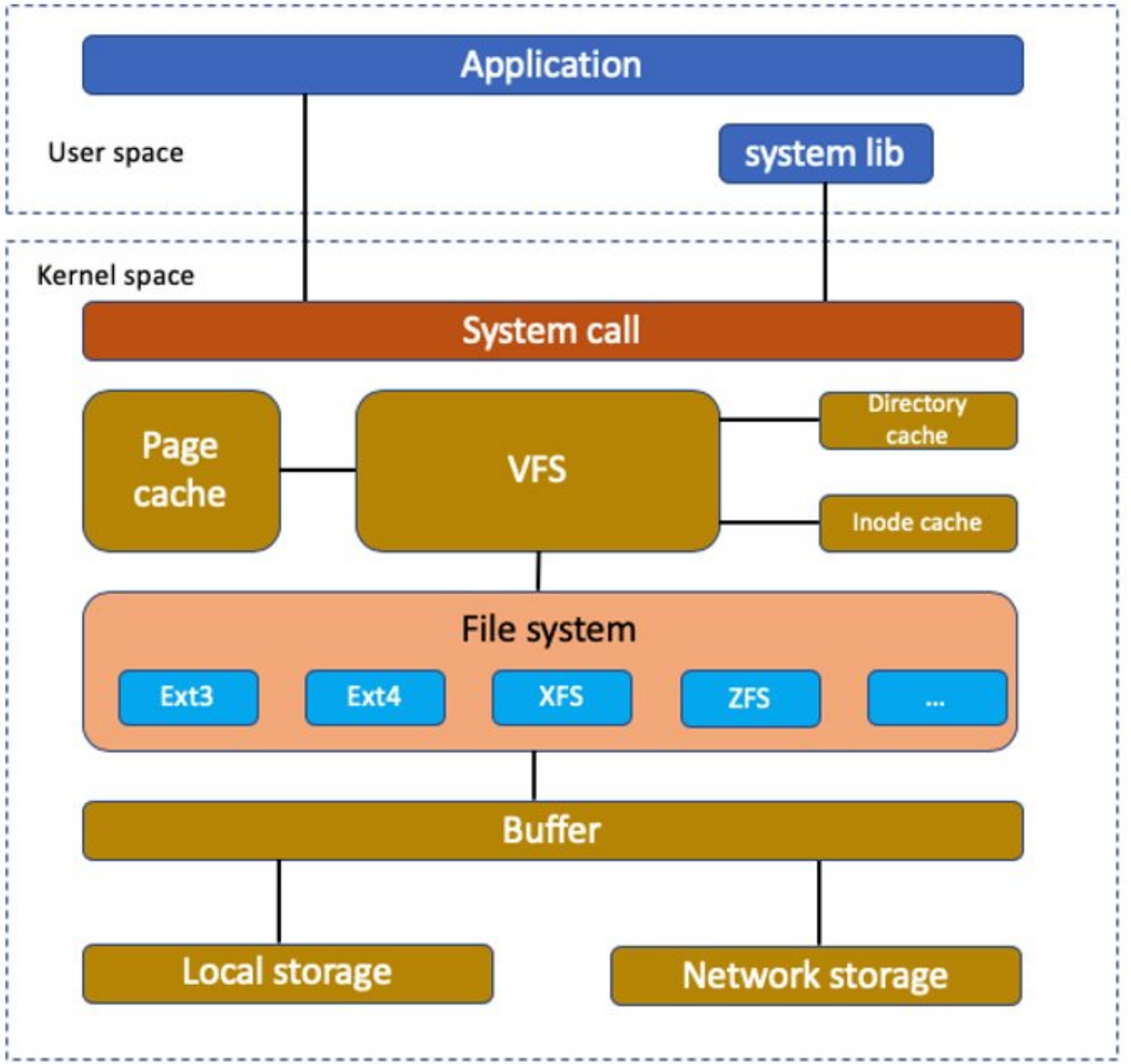


File Systems

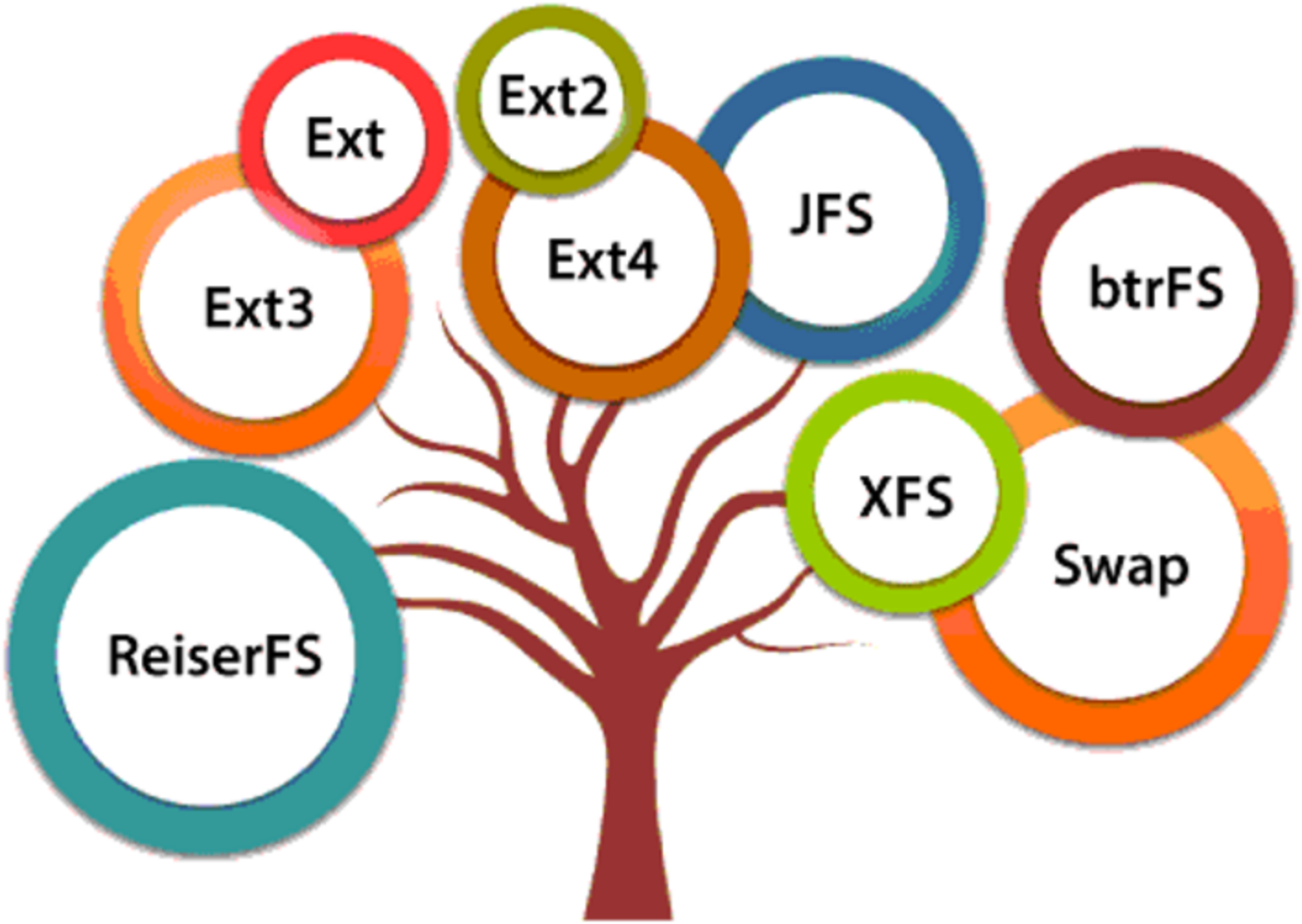
Linux File System Structure

The file system requires an API to access the function calls to interact with file system components like files and directories.

The first two parts of the given file system are called a **Linux virtual file system**. It provides a single set of commands for the kernel and developers to access the file system. This virtual file system requires the specific system driver to give an interface to the file system.



Types of Linux File Systems



File Systems

- **New Technology File System -** NTFS partitions can extend up to 16EiB Primarily used for Windows and Linux systems. Can't natively write to partition in MAC OS as it lacks the drives. However, third party tools can provide a way to do this.
- **File Allocation Table -** FAT is a general purpose file system that is compatible with Windows, Linux/Unix and MAC OS. Windows can't deal with anything greater than 32GB or FAT32. FAT suffers from over-fragmentation, file corruption, and limits to file names and size.

File Systems

- **Extended File Allocation Table -** exFAT is a Microsoft file system with compatibility with MAC OS 10.6+. Used in TVs and portable media players. Can be 512TBs in size. Doesn't work with Linux /Unix. Suffers from defragmentation and cannot pre-allocate disk space.
- **Hierarchical File System -** Built by Apple for MAC OS X. 8EiBs. Drivers are available for Linux systems to read and write to HFS+

File Systems

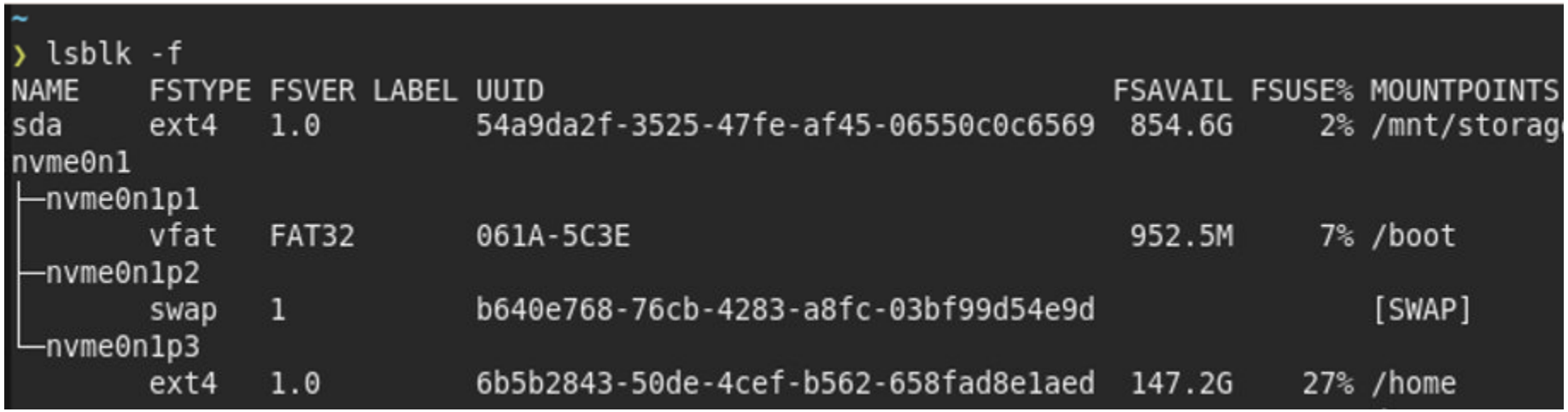
- **Extended File System -** EXT4, was created for Linux Kernels. Can be up to 1 EiB a file can be 16 TB. EXT4 is backward compatible with version 2 and 3. EXT4 can pre-allocated disk space. By default Windows and MAC OS cannon read EXT.
- **Journal File System -** A high-performance journaling file system, was first developed by Silicon Graphics for the IRIX operating system in 1993. It is the default file system for IRIX version 5.3 and then it was later ported to the Linux kernel.

File Systems

- **B-Tree File System -** Btrfs (b-tree fs), was created to address the lack of pooling, snapshots, checksums, and integrated multi-device spanning in Linux file systems, particularly as the need for such features emerged when working at the petabyte scale.
- **Zettabyte File System -** Merges the traditional volume management and filesystem layers, and it uses a copy-on-write transactional mechanism—both of these mean the system is very structurally different than conventional filesystems and RAID arrays

Small Computer System Interface Disk (scsi)

- The first hard disk detected on the SATA by the Linux port, system carries the label sda (sd => scsi disk)
- The second would be sdb
- nvme0n1 means first disk detected on the nvme port



Universally Unique Identifier

International Telecommunication Union (ITU) **ITU-T X.667 | ISO/IEC 9834-8**

UUIDs are an octet string of 16 octets (128 bits) in a 8-4-4-4-12 format:

- Hex: f81d4fae-7dec-11d0-a765-00a0c91e6bf6
- Dec: 329800735698586629295641978511506172918

Five versions:

- UUID1 = Timebased + Unique or MAC [no repeats till 3603AD]
- UUID2 = Timebased(LSB) + userid
- UUID3 = Namespace+MD5 hash
- UUID4 = PRNG [1 trillion UUIDs for a chance of 2 repeats]
- UUID5 = Namespace + SHA-1 hash

UUID4 Example

1. Generate 128 random bits:

   00000101 01100111 11010110 00011100 00101110 11100011 **10110010** 00111001 **00010100** 00010100 00010001 00010000 00100101 01101101 00100011 10000101

2. Take the 7th byte and perform an AND operation with `0x0F` to clear out the high nibble. Then, OR it with `0x40` to set the version number to 4.

   00000010 = **10110010** & 00001111

   10000010 = **00000010** | 10000000

UUID4 Example

3. Next, take the 9th byte and perform an AND operation with `0x3F` and then OR it with `0x80`.

$$00010100 = \mathbf{00010100} \ \& \ 00111111$$

$$10010100 = \mathbf{00010100} \ | \ 10000000$$

4. Convert the 128 bits to hexadecimal representation and insert the hyphens to achieve the canonical text representation.

567D61C2-EE30-2399-4141-110256D2385

Your Turn

10101110 00100001 10110100 11111100 01101111 01110010 **10100011** 00110010 **10111010** 10000110 11010010 00001001 11010001 11100000 10101111 01101001

2. Take the 7th byte and perform an AND operation with `0x0F` to clear out the high nibble. Then, OR it with `0x40` to set the version number to 4.
3. Next, take the 9th byte and perform an AND operation with `0x3F` and then OR it with `0x80`.
4. Convert the 128 bits to hexadecimal representation and insert the hyphens to achieve the canonical text representation.

▶ Answer

Everything is a File...

Inodes

- It is used to record the metadata of the file, such as the inode number, file size, access rights, modification date, data location, etc. One file has one associated inode, and like file content, inode stores on disk.

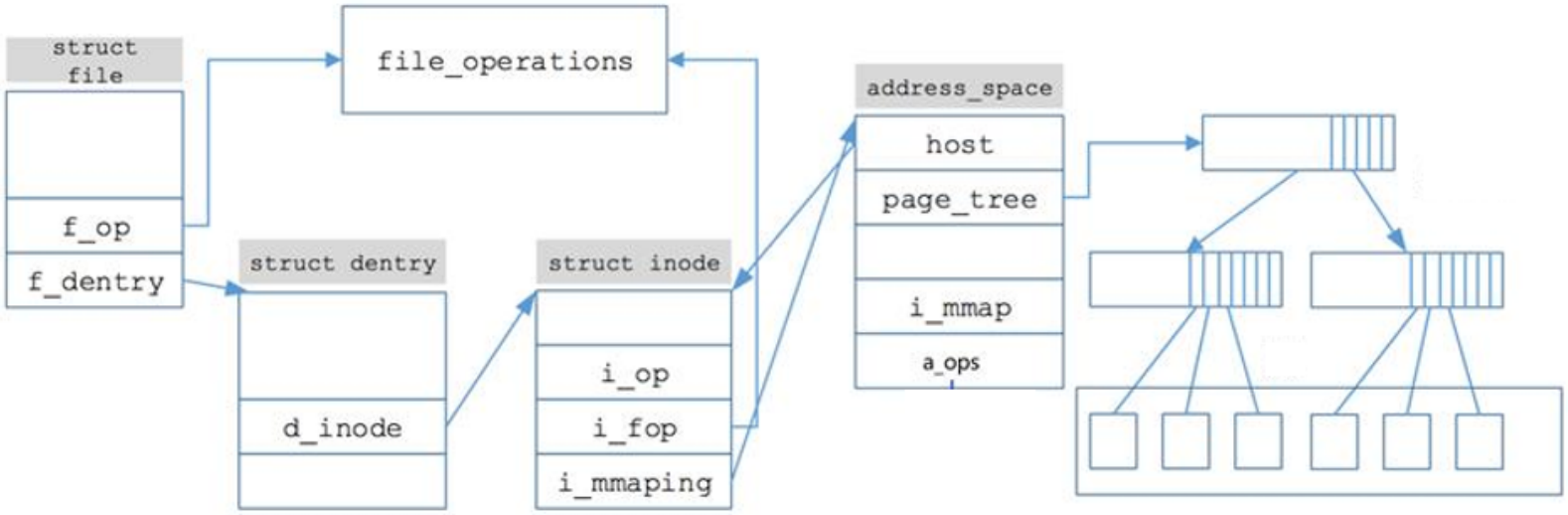
Directories

- D(d)entry for short, is used to record the name of the file, the inode pointer, and the association with other directory entries.
- Multiple associated directory entries constitute the directory structure of the file system.
- However, unlike inodes, a directory entry is an in-memory data structure maintained by the kernel, so it is often called a directory entry cache.

Superblock Information

```
❯ sudo dumpe2fs /dev/sda | grep -i superblock
dumpe2fs 1.46.5 (30-Dec-2021)
  Primary superblock at 0, Group descriptors at 1-117
  Backup superblock at 32768, Group descriptors at 32769-32885
  Backup superblock at 98304, Group descriptors at 98305-98421
  Backup superblock at 163840, Group descriptors at 163841-163957
  Backup superblock at 229376, Group descriptors at 229377-229493
  Backup superblock at 294912, Group descriptors at 294913-295029
  Backup superblock at 819200, Group descriptors at 819201-819317
  Backup superblock at 884736, Group descriptors at 884737-884853
  Backup superblock at 1605632, Group descriptors at 1605633-1605749
  Backup superblock at 2654208, Group descriptors at 2654209-2654325
  Backup superblock at 4096000, Group descriptors at 4096001-4096117
  Backup superblock at 7962624, Group descriptors at 7962625-7962741
  Backup superblock at 11239424, Group descriptors at 11239425-11239541
  Backup superblock at 20480000, Group descriptors at 20480001-20480117
  Backup superblock at 23887872, Group descriptors at 23887873-23887989
  Backup superblock at 71663616, Group descriptors at 71663617-71663733
  Backup superblock at 78675968, Group descriptors at 78675969-78676085
  Backup superblock at 102400000, Group descriptors at 102400001-102400117
  Backup superblock at 214990848, Group descriptors at 214990849-214990965
```

File, Inode, Address_Space




```
struct address_space_operations{
    int (*writepage)(struct page *page, struct writeback_control *wbc);
    int (*readpage)(struct file *, struct page *);
    /* write back some dirty pages from this mapping */
    int (*writepages)(struct address_space *, struct writebackcontrol *);
    /* set a page dirty. Return true if this dirited it */
    int (*set_page_dirty)(stuct page *page)
    int (*readpages)(struct, file *filp, struct address_space *mapping,
                                struct list_head *pages, unsigned nr_pages);
    …
}
```

Strace

Tracing system calls and signals

```
❯ strace cat file1
execve("/usr/bin/cat", ["cat", "file1"], 0x7ffc04c28a98 /* 40 vars */) = 0
brk(NULL)                               = 0x5620c1da0000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd7148a490) = -1 EINVAL (Invalid argument)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=94735, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 94735, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f441c3de000
close(3)                                = 0
openat(AT_FDCWD, "/usr/lib/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\320\324\2\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0@\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0"..., 80, 848) = 80
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\205vn\235\204X\261n\234|\346\340|q,\2"..., 68, 928) = 68
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2463384, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f441c3dc000
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
mmap(NULL, 2136752, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f441c1d2000
mprotect(0x7f441c1fe000, 1880064, PROT_NONE) = 0
mmap(0x7f441c1fe000, 1531904, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2c000) = 0x7f441c1fe000
mmap(0x7f441c374000, 344064, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1a2000) = 0x7f441c374000
mmap(0x7f441c3c9000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1f6000) = 0x7f441c3c9000
mmap(0x7f441c3cf000, 51888, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f441c3cf000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f441c1cf000
arch_prctl(ARCH_SET_FS, 0x7f441c1cf740) = 0
set_tid_address(0x7f441c1cfa10)         = 223554
set_robust_list(0x7f441c1cfa20, 24)     = 0
rseq(0x7f441c1d00e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7f441c3c9000, 12288, PROT_READ) = 0
mprotect(0x5620c1149000, 4096, PROT_READ) = 0
mprotect(0x7f441c42b000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f441c3de000, 94735)           = 0
getrandom("\xb4\x9e\x9d\x51\x0c\xd0\x6b\x2e", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x5620c1da0000
brk(0x5620c1dc1000)                     = 0x5620c1dc1000
openat(AT_FDCWD, "/usr/lib/locale/locale-archive", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=3048976, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 3048976, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f441bee6000
close(3)                                = 0
newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}, AT_EMPTY_PATH) = 0
openat(AT_FDCWD, "file1", O_RDONLY)     = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=6, ...}, AT_EMPTY_PATH) = 0
fadvise64(3, 0, 0, POSIX_FADV_SEQUENTIAL) = 0
mmap(NULL, 139264, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f441bec4000
read(3, "Hello\n", 131072)              = 6
write(1, "Hello\n", 6Hello
)                 = 6
read(3, "", 131072)                     = 0
munmap(0x7f441bec4000, 139264)          = 0
close(3)                                = 0
close(1)                                = 0
close(2)                                = 0
exit_group(0)                           = ?
```