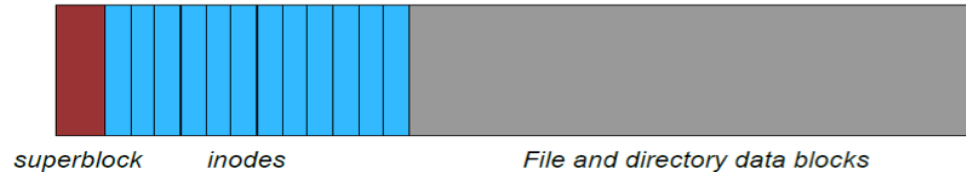


Reminder: Model disk layout for an ext2 filesystem. Inodes on disk have pointers to ~10 direct disk block entries, one indirect, one double indirect, one triple indirect block.



For ext2 with 4KB blocks and 32 bit addressing. What is the maximum supported disk size in bytes?

Each inode entry is 128 bytes and during formatting 64KB is reserved for inode array. How many files and directories can you create?

For ext2 filesystem with 4KB blocks and 32 bit addressing, how large can a file be before a triple indirect block is required?

Big idea: Forget names of files: *The 'inode' is the file.*

What does the following do?

```
char mystery[PATH_MAX+1];
if (getcwd(mystery, PATH_MAX)) puts(mystery);?
```

How do we implement a directory? (spot the mistake)

```
DIR* dirp = opendir(".");
while ((dp = readdir(dirp)) != NULL) {
    puts(dp->d_name);
    if (!strcmp(dp->d_name, name)) {
        return 1; /* Found */
    }
}
closedir(dirp);
return 0; /* Not Found */
```

How can I find the inode number of a file?

How do I find out meta- file information?

```
int stat(const char *path, struct stat *buf);
int fstat(int fd, struct stat *buf);
int lstat(const char *path, struct stat *buf);
```

```
struct stat {
    dev_t      st_dev;      ID of device containing file
    ino_t      st_ino;      inode number
    mode_t     st_mode;     protection & other info
    nlink_t    st_nlink;    number of hard links
    uid_t      st_uid;      user ID of owner
    gid_t      st_gid;      group ID of owner
    dev_t      st_rdev;     device ID (if special file)
    off_t      st_size;     total size, in bytes
    blksize_t  st_blksize;  blocksize for file system
    blkcnt_t   st_blocks;   number of 512B blocks allocated
    time_t     st_atime;    time of last access
    time_t     st_mtime;    time of last modification
    time_t     st_ctime;    time of last status change
};
```

Users are integers!?

st_mtime vs st_ctime?

mymake.c ; compile iff source code is newer or target does not exist?

```
int s_ok = stat("prog.c", &src);
int t_ok = stat("a.out", &tgt);

double delta = difftime(_____, _____)
// -ve if t1 before t2

if(_____) {
    puts("Compiling");
    system("gcc prog.c"); // = fork, exec shell, wait
} else { puts("nothing to do"); }
```

Does the inode contain the filename [10¹⁰ points]?

How can I have the same file appear in two different places in my file system?

(From code? Command?)

Reference counting?

rm = unlinking?

Changing File Permissions?

```
chmod 644 /bin/sandwhich
```

```
chmod 755 /bin/sandwhich
```

```
chmod ugo-w /bin/sandwhich
```

```
chmod o-rx /bin/sandwhich
```

From code ... `chmod(const char *path, mode_t mode);`

What are the two "set uid bits" ?

set-user-ID-on-execution/set-group-ID-on-execution

Why are they useful? What common linux program uses this feature?

ext3: Journaling. Able to rollback to a known good state.

ext4: Performance. Encryption. Better limits (e.g. #files per dir)

Case study: ext4 has the "delayed data-write problem"

```
fd=open("file", O_TRUNC); write(fd, data); close(fd);
```

```
fd=open("file.tmp");
```

```
write(fd, data);
```

```
close(fd);
```

```
rename("file.tmp", "file"); // Very happy in ext3
```

// but upgrading to ext4 : the rename could be completed before content is written to disk surface!

ZFS

1.1 Data integrity

1.2 RAID

1.3 Storage pools

1.4 ZFS cache: ARC (L1), L2ARC, ZIL

1.5 Gigantic Capacity (128bit model)

1.6 Copy-on-write transactional model

1.7 Snapshots and clones

1.8 Sending and receiving snapshots

1.9 Dynamic striping

1.10 Variable block sizes

1.11 Lightweight filesystem creation

1.12 Cache management

1.13 Adaptive endianness

1.14 Deduplication

1.15 Encryption

BtrFS

Extent based file storage

2⁶⁴ byte == 16 EiB maximum file size (practical limit is 8 EiB due to Linux VFS)

Space-efficient packing of small files

Space-efficient indexed directories

Dynamic inode allocation

Writable snapshots, read-only snapshots

Subvolumes (separate internal filesystem roots)

Checksums on data and metadata (crc32c)

Compression (zlib and LZO)

Integrated multiple device support

File Striping, File Mirroring, File Striping+Mirroring, Striping with Single and Dual Parity implementations

SSD (Flash storage) awareness (TRIM/Discard for reporting free blocks for reuse) and optimizations

Efficient Incremental Backup

Background scrub process for finding and fixing errors on files with redundant copies

Online filesystem defragmentation

Offline filesystem check

In-place conversion of existing ext3/4 file systems

Seed devices. Create a (readonly) filesystem that acts as a template to seed other Btrfs filesystems. The original filesystem and devices are included as a readonly starting point for the new filesystem. Using copy on write, all modifications are stored on different devices; the original is unchanged.

Subvolume-aware quota support

Send/receive of subvolume changes

Efficient incremental filesystem mirroring

Batch, or out-of-band deduplication (happens after writes, not during)