File Systems: V6 Case study

- Learning Objective
  - Describe the UNIX V6 file system.
  - Identify the strengths and weaknesses of the V6 file system.
  - Why V6: simple, but ancestor of most modern file systems.
- Topics:
  - Overview of the V6 file system
    - Disk representation
    - Directory structure
    - Recovery characteristics

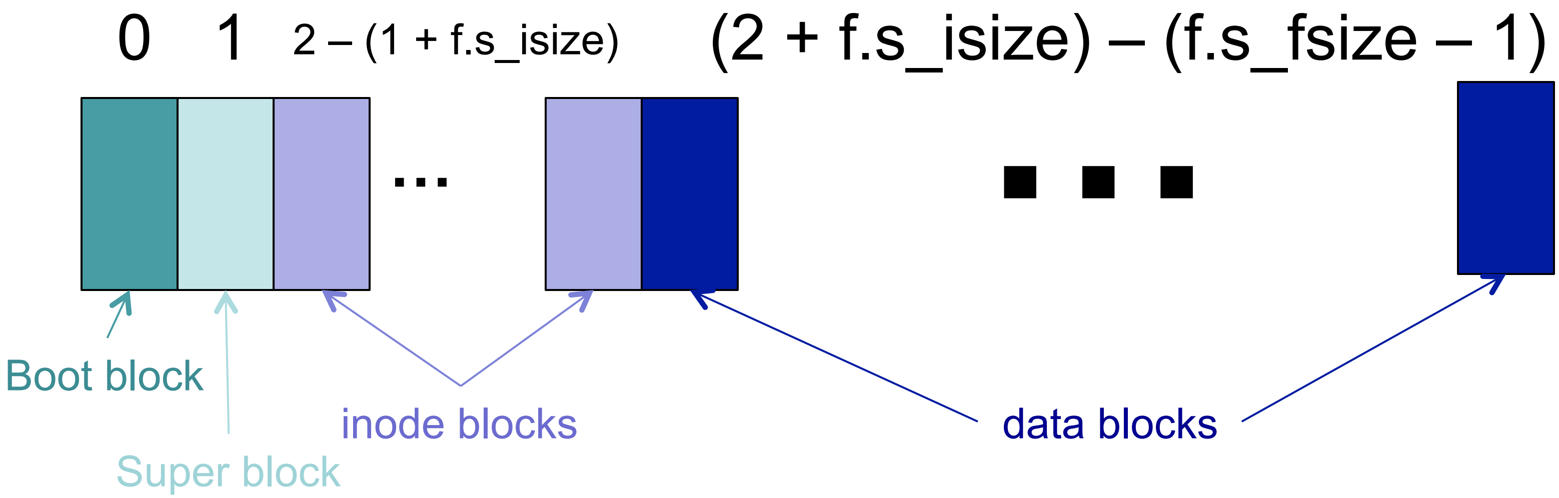
- With enormous thanks to:
  - Ken Thompson and Dennis Ritchie
  - John Lions (https://en.wikipedia.org/wiki/Lions'_Commentary_on_UNIX_6th_Edition,_with_Source_Code)
  - Keith Bostic
  - Whoever is behind: http://v6.cuzuco.com/

Overview

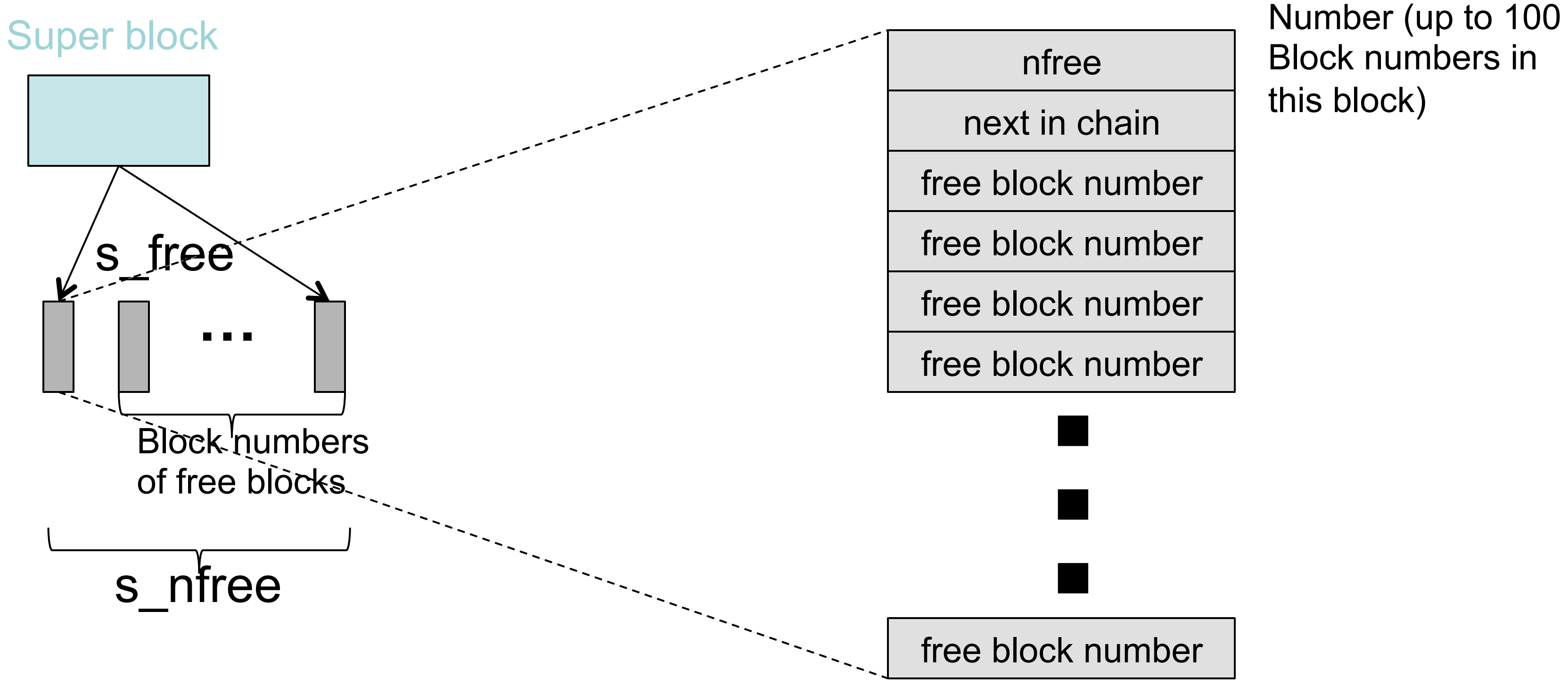
- Disk is divided up into 512-byte blocks
  - Block 0: boot block
  - Block 1: superblock (struct filsys)
  - Blocks 2 – f(Ninodes): inodes (16 inodes/block)
  - Rest of disk contains file data (and spare blocks for “bad block” handling)
- Free Space management
  - Up to 99 blocks, referenced directly in the superblock.
  - 1 block as the head of a linked list of blocks containing addresses of other free blocks (pictures coming).

Disk Layout

Block numbers



Free Space Management



Super block

s_free

…

Block numbers of free blocks

s_nfree

nfree

next in chain

free block number

free block number

free block number

free block number

■

■

■

free block number

Number (up to 100 Block numbers in this block)

File system level metadata

- `struct filsys`
  - Today, we call this the superblock
  - Created when you create the file system
  - Read when you mount the file system

```
struct filsys {
 int s_isize;        /* size in blocks of the I list */
 int s_fsize;        /* size in blocks of the entire volume */
 int s_nfree;        /* number of in core free blocks
                        (between 0 and 100) */
 int s_free[100];    /* in core free blocks */
 int s_ninode;       /* number of in core I nodes (0-100) */
 int s_inode[100];   /* in core free I nodes */
 char s_flock;       /* lock during free list manipulation */
 char s_ilock;       /* lock during I list manipulation */
 char s_fmod;        /* super block modified flag */
 char s_ronly;       /* superb lock modified flag */
 int s_time[2];      /* current date of last update */
 int pad[50];
}
```

Per-file Metadata (on-disk)

- On disk inode:

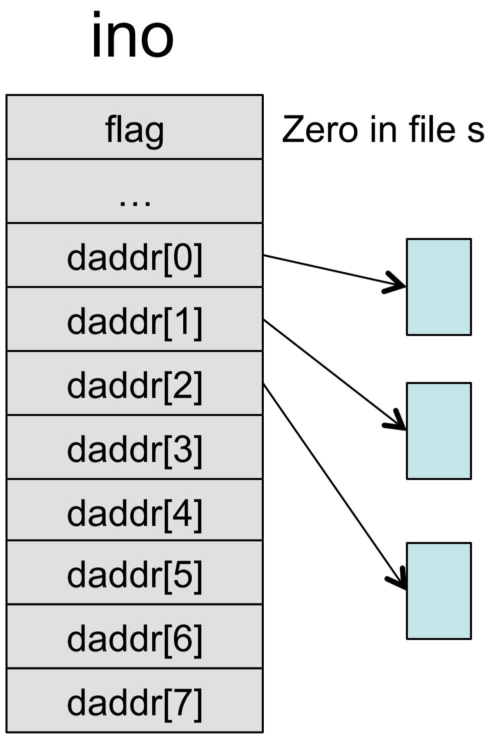
```
struct ino {
    int     i_mode;         /* File mode */
    char    i_nlink;        /* Link count */
    char    i_uid;          /* Owner user id */
    char    i_gid;          /* Group id */
    char    i_size0;        /* most significant of size */
    char    *i_size1;       /* least sig */
    int     i_addr[8];      /* Disk addresses of blocks */
    int     i_atime[2];     /* Access time */
    int     i_mtime[2];     /* Modified time */
}
```

Per-file Metadata (in-memory)

- In-memory inode:

```
struct inode {
    char    i_flag;
    char    i_count;       /* reference count */
    int     i_dev;         /* device where inode resides */
    int     i_number;      /* i number 1:1 w/device addr */
    int     i_mode;
    char    i_nlink;       /* directory entries*/
    char    i_uid;         /* owner */
    char    i_gid;         /* group of owner */
    char    i_size0;       /* most significant of size */
    char    *i_size1;      /* least sig */
    int     i_addr[8];     /* device addresses constituting file */
    int     i_lastr;       /* last logical block read */
}
```

Different sized files (1)



ino

flag

…

daddr[0]

daddr[1]

daddr[2]

daddr[3]

daddr[4]

daddr[5]

daddr[6]

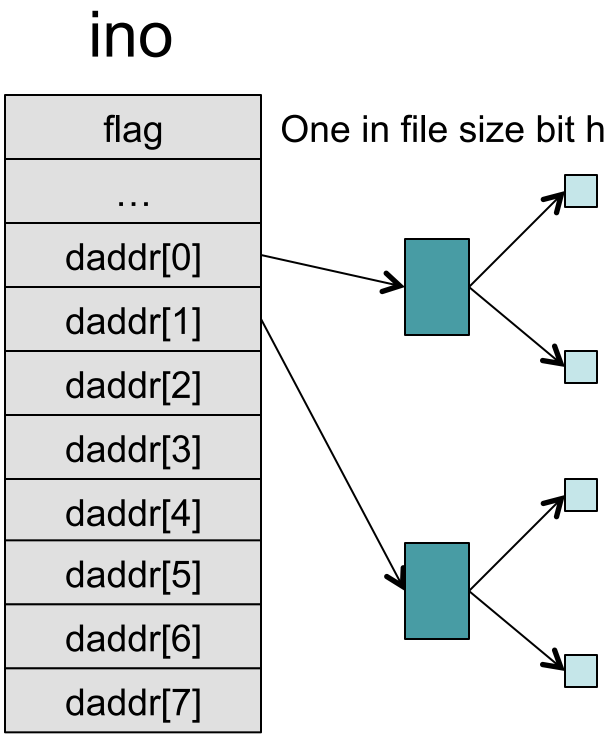
daddr[7]

Zero in file size bit here indicates “small” file

Small file:

- 8 or fewer blocks.
- Daddr[0-7] contain addresses of data blocks.
- Max size:
  - 512 * 8 = 4 KB

Different sized files (2)



ino

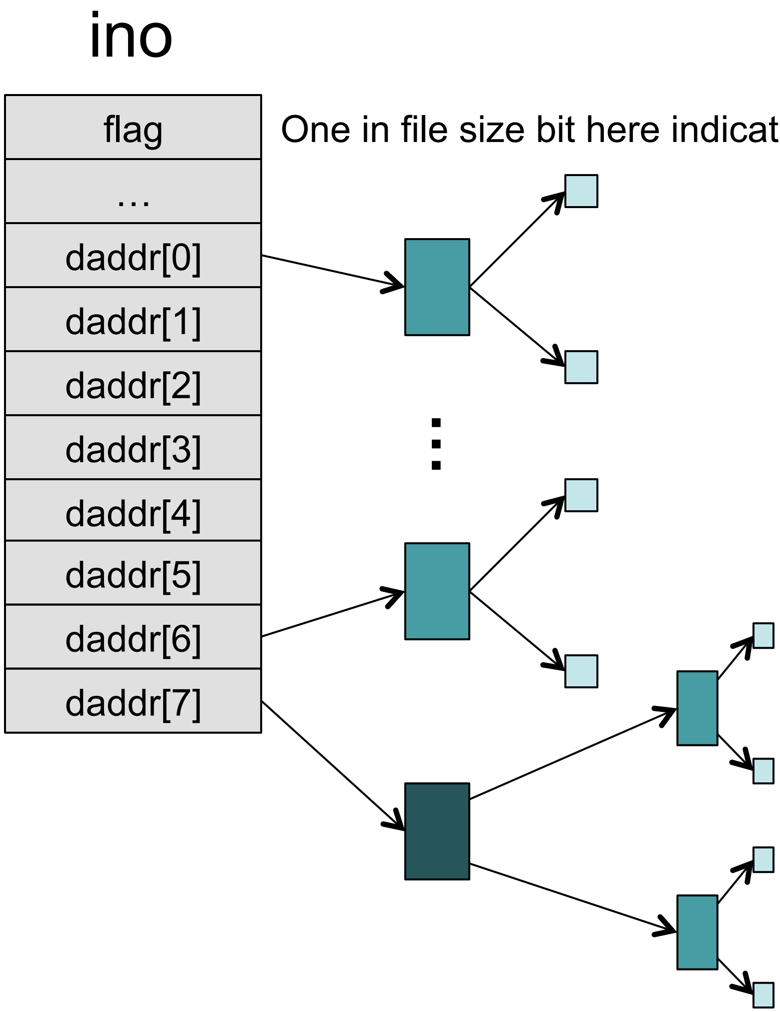
ino
flag
…
daddr[0]
daddr[1]
daddr[2]
daddr[3]
daddr[4]
daddr[5]
daddr[6]
daddr[7]

One in file size bit here indicates “large” file

Large file:

- Daddr[0-] contain addresses of indirect blocks.
- Daddr[7] contains the address of a doubly indirect block.
- Assume that daddr[7] is 0 for now.
- Max size:
  - 7 * 256 * 512 = 896 KB

Different sized files (3)



ino

flag

…

daddr[0]

daddr[1]

daddr[2]

daddr[3]

daddr[4]

daddr[5]

daddr[6]

daddr[7]

One in file size bit here indicates “large” file

Huge file:

- 7 indirect blocks addresses in Daddr, AND
- Daddr[8] contains a doubly indirect block.
- Max size:
  - 7 * 256 * 512 = 896 KB
  - 256 * 256 * 512 = 32 MB
  - MAX = 32 MB + 896 KB

Directory Entries

- Hierarchical directory structure that you know and love, including “.” and “..”.
- Directory entries are 16 bytes:
  - 2 bytes of inode number
  - 14 bytes (right padded) of name
- A directory entry with inode = 0 is unused

Exercise

- Critique this file system design:
  1. How well does it handle sequential access?
  2. How well does it handle random access?
  3. What kinds of free space management problems can you foresee?
  4. Would this work well for a system of many tiny files?
  5. How about a system of many huge files?
  6. What kinds of errors might arise if you crash with dirty data buffered in memory?
  7. What would you do while the system is running to protect against such errors?