PERSISTENCE: FILE SYSTEMS

Kai Mast

CS 537

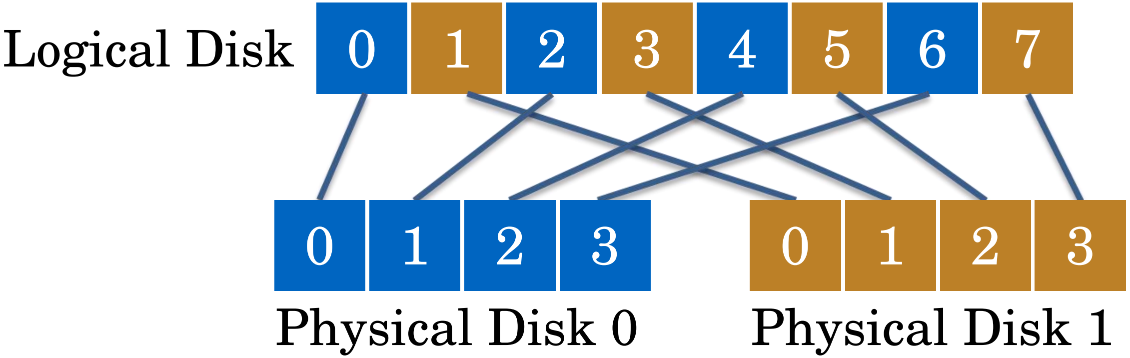
Fall 2022

ANNOUNCEMENTS

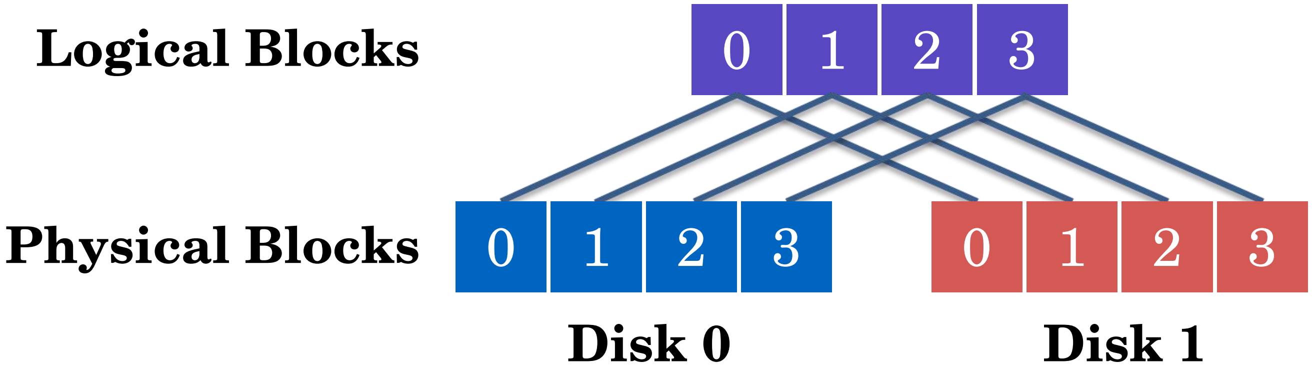
- No office hours on Wednesday and Thursday (Thanksgiving)
  - You get free slip days for those
  - The latest you can submit P3b (with slip days) is next Saturday
- P4a will be released on Monday 11/28
  - Will be the last project
- See note on Piazza on what to do before asking questions in OH

RAID-0: STRIPING

Optimizes for capacity. Does not provide redundancy.

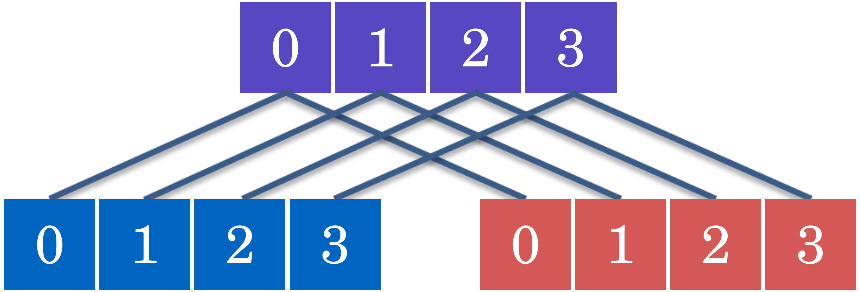


RAID-1: MIRRORING



Keep two copies of every block

RAID-1: MIRRORING



How many disks can fail without losing any data?

RAID-1 can always handle 1 disk failure

RAID-1 WITH STRIPING

Disk 0	Disk 1	Disk 2	Disk 3
0	0	1	1
2	2	3	3
4	4	5	5
6	6	7	7

Combines mirroring and striping

- Capacity is N*C/2
- Can handle at least 1 and up to N/2 failures
    - e.g., 1 and 3 can fail, but not 2 and 3
- Also called RAID 10 or RAID 1+0

RAID-1: ANALYSIS

What is the total capacity?	**N/2 * C**
How many disks can fail?	**1 (or maybe N / 2)**
Read Latency?	**D**
Write Latency?	**D**

N := number of disks
C := capacity of 1 disk
S := sequential throughput of 1 disk
R := random throughput of 1 disk
D := latency of one small I/O operation

Disk 0	Disk 1	Disk 2	Disk 3
0	0	1	1
2	2	3	3
4	4	5	5
6	6	7	7

RAID-1: THROUGHPUT

What is the steady-state throughput for

- random reads? **N * R**
- random writes? **N/2 * R**
- sequential writes? **N/2 * S**
- sequential reads? **N/2 * S**

If a fully sequential read is split across all four disks, each disk would spend half its time spinning to the next location

Disk 0	Disk 1	Disk 2	Disk 3
0	0	1	1
2	2	3	3
4	4	5	5
6	6	7	7

SIDE ISSUE: SYSTEM CRASHES

RAID 1

Block	Disk 0	Disk 1
0	A	A
1	X	B
2	C	C
3	D	D

System crashes can happen due to bugs or power loss

- Requires reboot!

Application writes “X” to block 1

- Disk 0 writes to block 2 successfully
- System crashes before Disk 1 is done writing to block 2

Problem: After reboot, how to tell which data is right?

CRASHES: H/W SOLUTION

Consistent-Update Problem:
We want writes on both/all disk to be atomic

Solution: Use non-volatile RAM in RAID controller

- Can replay to ensure all copies are updated
- Software RAID controllers (e.g., Linux md) don't have this option

RAID-4 STRATEGY

RAID-4: Compromise between RAID-0 and RAID-1

Use **one** disk for **parity** (form of redundancy, but not full replication)

In algebra: Equation with N variables and N-1 are known, can often solve for unknown

Treat sectors across disks in a stripe as equation

Data on bad disk is the unknown in equation

RAID-4 WITH PARITY

Disk 0	Disk 1	Disk 2	Disk 3
0	1	2	P0
3	4	5	P1
6	7	8	P2
9	10	11	P3

- **Data** blocks on disks 0, 1, and 2
- Disk 3 for **parity**
- Parity calculated over data blocks in stripe

$\text{Parity}_0 = \text{Data}_0 \textbf{ XOR } \text{Data}_1 \textbf{ XOR } \text{Data}_2$

Parity Example: 1

	Disk0	Disk1	Disk2	Disk3	Disk4
Stripe:	010	011	101	111	011
					(parity)

Calculate (even) parity from data blocks

$\text{Parity}_0 = \text{Data}_0 \textbf{ XOR } \text{Data}_1 \textbf{ XOR } \text{Data}_2 \textbf{ XOR } \text{Data}_3$

Parity Example: 1

	Disk0	Disk1	Disk2	Disk3	Disk4
Stripe:	010	011	101	111	011
					(parity)

Can reconstruct blocks of lost disk by taking XOR

$\mathrm{Data}_1 = \mathrm{Data}_0$ **XOR** Data_2 **XOR** Parity_0

UPDATING PARITY: XOR

If write “0110” to block 0, how should parity be updated?
(assume current value is 1100)

Slow approach: read all other N-2 blocks in stripe and calculate new parity

Faster approach

- Read old value at block 0, then XOR with new data: 1100 XOR 0110
- Read old value for parity, then XOR with the above XOR’ed value
  0101 XOR (1100 XOR 0110)
- Calculate new parity: 1111
- Write out new parity: 2 reads and 2 writes (1 read and 1 write to parity block)

RAID-4: ANALYSIS

What is the total capacity?	**(N-1) * C**
How many disks can fail?	**1**
Read Latency?	**D**
Write Latency?	**2*D (read and write parity disk)**

N := number of disks
C := capacity of 1 disk
S := sequential throughput of 1 disk
R := random throughput of 1 disk
D := latency of one small I/O operation

Disk 0	Disk 1	Disk 2	Disk 3
0	1	2	P0
3	4	5	P1
6	7	8	P2
9	10	11	P3

RAID-4: THROUGHPUT

How to avoid the parity bottleneck?

What is steady-state throughput for

- sequential reads? **(N-1) * S**
- sequential writes? **(N-1) * S (parity calculated for full stripe)**
- random reads? **(N-1) * R**
- random writes? **R/2 (read and write parity disk)**

N := number of disks
C := capacity of 1 disk
S := sequential throughput of 1 disk
R := random throughput of 1 disk
D := latency of one small I/O operation

Disk 0	Disk 1	Disk 2	Disk 3
0	1	2	P0
3	4	5	P1
6	7	8	P2
9	10	11	P3

RAID-5

Disk0	Disk1	Disk2	Disk3	Disk4
-	-	-	-	P
-	-	-	P	-
-	-	P	-	-

…

Rotate parity across different disks

Where exactly do individual data blocks go?

RAID-5

Disk0	Disk1	Disk2	Disk3	Disk4
0	1	2	3	**P0**
5	6	7	**P1**	4
10	11	**P2**	8	9
15	**P3**	12	13	14
P4	16	17	18	19

Pattern repeats…

Sector number get shifted left (or right)

RAID-5: ANALYSIS

What is the total capacity? **(N-1) * C**
How many disks can fail? **1**
Read Latency? **D**
Write Latency? **2*D (read and write parity disk)**

These metrics same as RAID-4…

N := number of disks
C := capacity of 1 disk
S := sequential throughput of 1 disk
R := random throughput of 1 disk
D := latency of one small I/O operation

Disk0	Disk1	Disk2	Disk3	Disk4
-	-	-	-	P
-	-	-	P	-
-	-	P	-	-

…

RAID-5: Throughput

Steady-state throughput for RAID-4

- sequential reads? **(N-1) * S**
- sequential writes? **(N-1) * S (parity calculated for full stripe)**
- random reads? **(N-1) * R**
- random writes? **R/2 (read and write parity disk)**

What is steady-state throughput for RAID-5?

- sequential reads? **(N-1) * S**
- sequential writes? **(N-1) * S**
- random reads? **N * R**
- random writes? **N * R/4 (2 read and 2 writes per logical write)**

RAID LEVEL COMPARISONS

	Reliability	Capacity
RAID-0	0	C*N
RAID-1	1	C*N/2
RAID-4	1	(N-1) * C
RAID-5	1	(N-1) * C

Other RAID levels exist, but are not covered in this course

RAID Level Comparisons

	Read Latency	Write Latency
RAID-0	D	D
RAID-1	D	D
RAID-4	D	2D
RAID-5	D	2D

RAID LEVEL COMPARISONS

	Sequential Read	Sequential Write	Random Read	Random Write
RAID-0	N * S	N * S	N * R	N * R
RAID-1	N/2 * S	N/2 * S	N * R	N/2 * R
RAID-4	(N-1)*S	(N-1)*S	(N-1)*R	R/2
RAID-5	(N-1)*S	(N-1)*S	N * R	N/4 * R

RAID-5 is strictly better than RAID-4

RAID-0 is always fastest and has best capacity (but at cost of reliability)

RAID-1 better than RAID-5 for random workloads

RAID-5 better than RAID-1 for sequential workloads

File System API

(Book Chapter 39)

WHAT IS A FILE?

Array of persistent bytes that can be read/written

Two interpretations of "file system"

1. Collection of files (**file system image**)
2. Part of OS that manages those files
    - Many local file systems: ext2, ext3, ext4, xfs, zfs, btrfs, f2fs
    - Files are common abstraction across all…

Files need **names** so can they can be addressed/accessed properly

Three types of names

1. Unique id: inode numbers
2. Path
3. File descriptor

1) NAME: INODE NUMBER

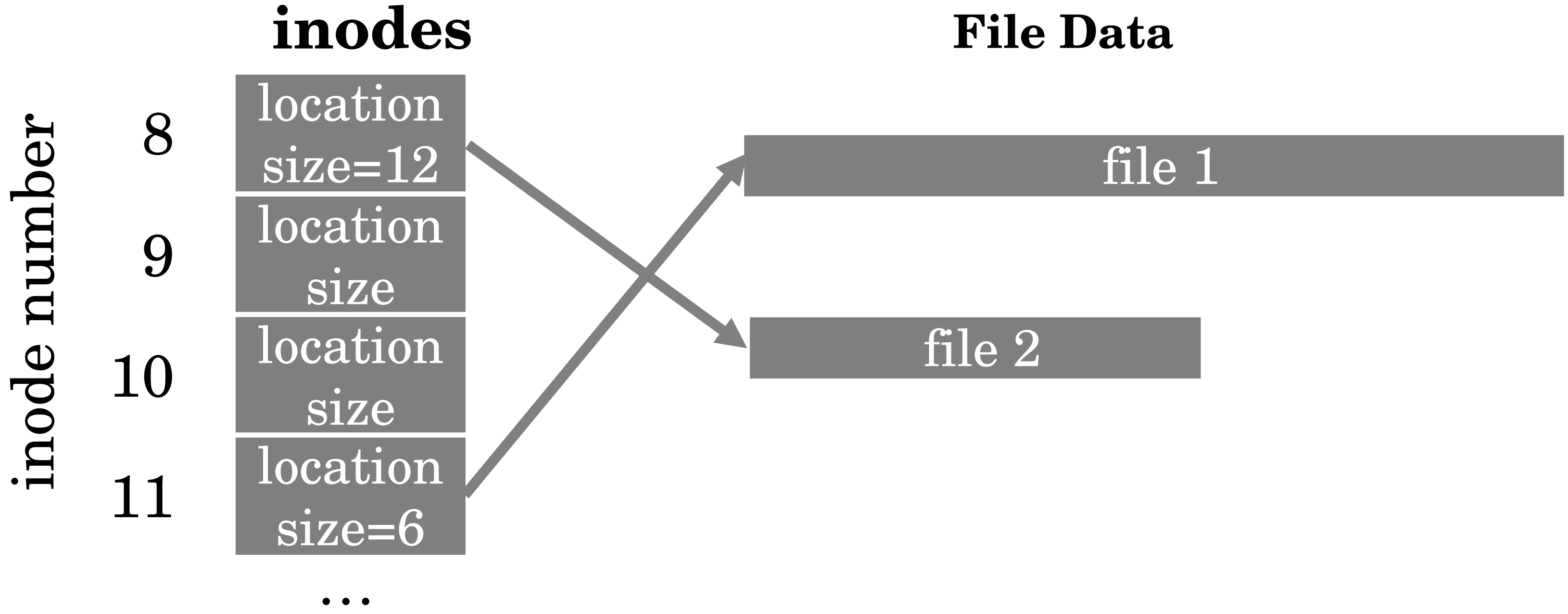
Each file has exactly one **inode number**

Inodes are unique (at a given time) within file system image

Different file systems may use the same number,
Numbers may be recycled after deletes

See inodes via "ls –i"; (see them increment as create new files…)

```
CS537> ls -li
total 38328
4195226 -rw-rw-r-- 1 kai kai 1219902 Sep 13 09:15 01-Introduction.pdf
4195155 -rw-r--r-- 1 kai kai 2185784 Sep 13 09:15 01-Introduction.pptx
[...]
```



Meta-data: Describes data

- We will discuss file meta-data more in next lecture

Often, inodes are stored in known, fixed block location on disk

- Simple arithmetic determines location of particular inode

FILE API (ATTEMPT 1)

```
read(int inode, void *buf, size_t nbyte)
write(int inode, void *buf, size_t nbyte)
seek(int inode, off_t offset)
```

Common case: sequential accesses
`read()` and `write()` track current offset of file to access next

Read/write random location in file:
`seek()` sets offset; does not cause disk seek until read/write performed

Disadvantages

- names hard to remember
- no organization or meaning to inode numbers
- semantics of offset across multiple processes?

2) PATHS

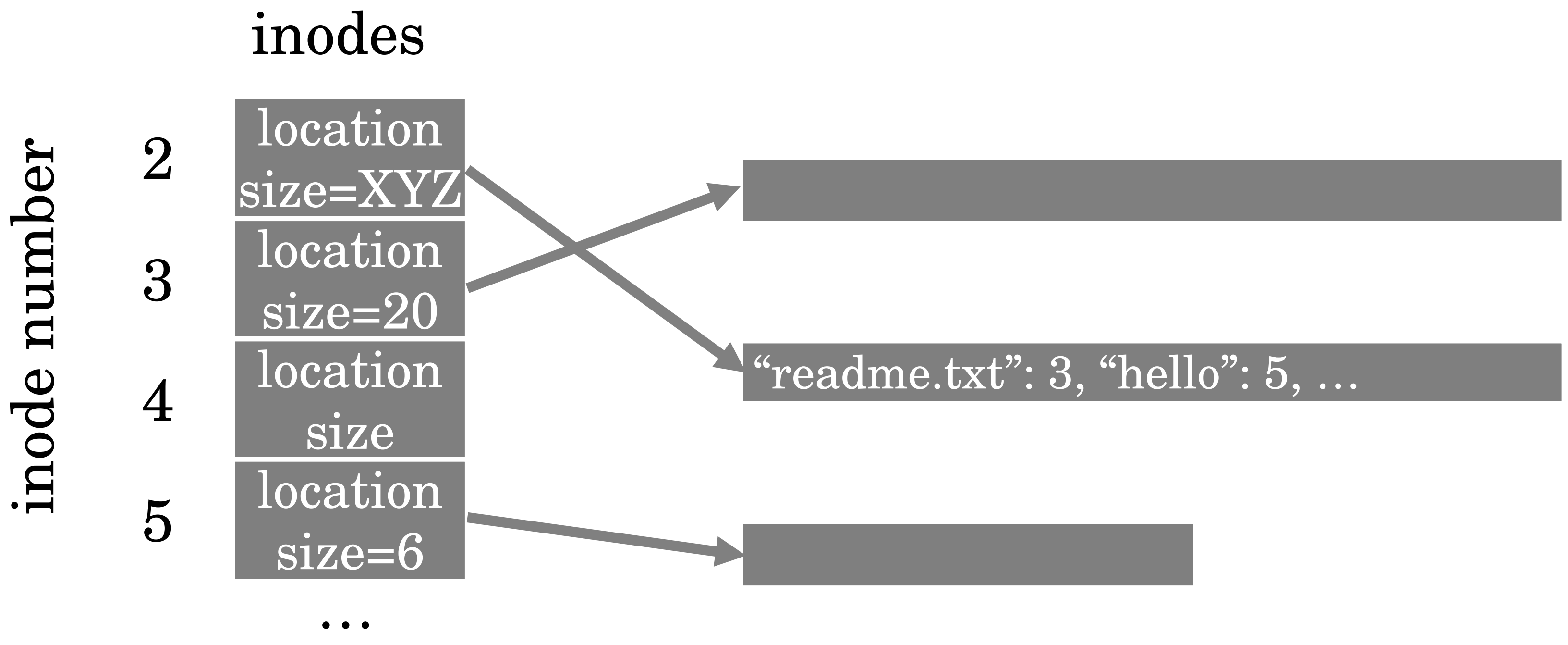
String names are better for users than numbers

File system still interacts with inode numbers

Store *path-to-inode* mappings in special files; what is that special file?

Directory!

Start with a single directory, stored in known location (root directory, typically inode 2)



What does inode number 2 point to?

What is the name of the file stored with inode 3? File with inode 5?

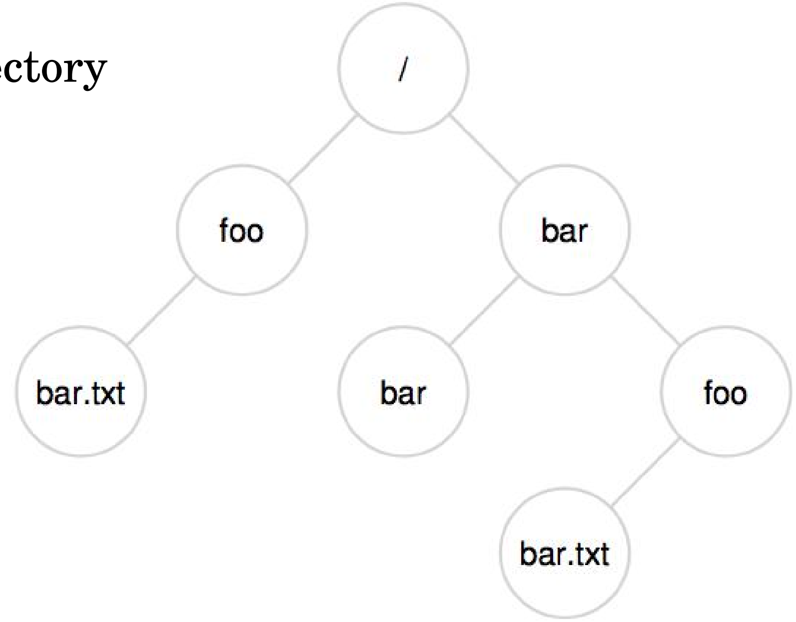
2) PATHS

Generalize to multiple directories…

Directory Tree instead of single root directory
File name needs to be unique only within a directory

What are the path names of all the files?

```
/foo/bar.txt
/bar/bar
/bar/foo/bar.txt
```

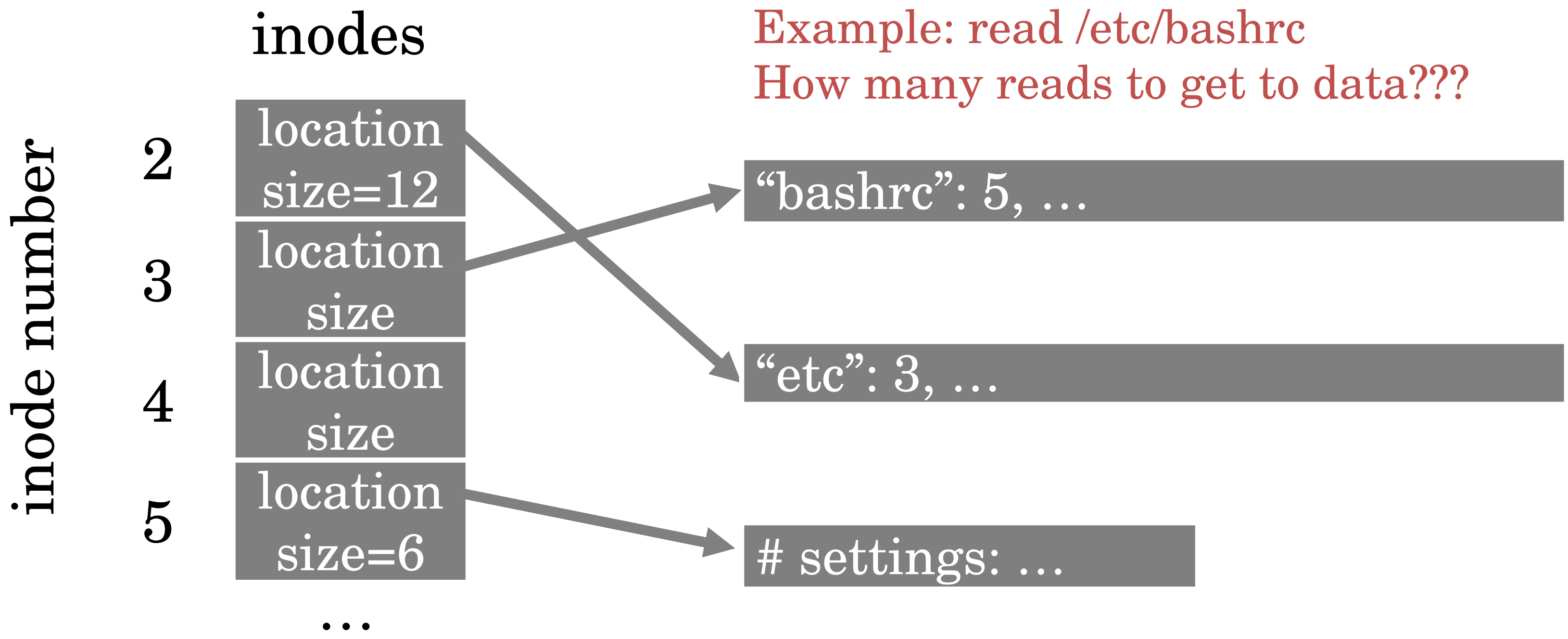


Store file-to-inode mapping in each directory

Reads for getting final inode called "traversal"

Example: read /etc/bashrc

How many reads to get to data???



1. Inode #2 Get location of root directory
2. Read root directory data; see “etc” maps to inode 3
3. Inode #3 Get location of etc directory
4. Read /etc directory; see “bashrc” is at inode 5
5. Inode #5 Get location of /etc/bashrc file data
6. Read /etc/bashrc file data

How to create /etc/newfile?

DIRECTORY CALLS

Directories are stored very similarly to files

Add a bit to inode to designate if data is for "file" or "directory"

`mkdir`: create new directory

`readdir`: read/parse directory entries

No `writedir`, instead:

- `create`/`open`, `unlink` files inside the directory
- `mkdir` and `unlink` subdirectories
- `unlink` to delete the directory itself

SPECIAL DIRECTORY ENTRIES

Output of cd `/; ls -lia`

```
 prompt /> ls -lia total 66
       2 drwxr-xr-x  18 root root  4096 Oct 21 13:07 ./
       2 drwxr-xr-x  18 root root  4096 Oct 21 13:07 ../
40894465 drwxr-xr-x   2 root root  4096 Aug 20 16:48 afs/
      12 lrwxrwxrwx   1 root root     7 Oct 18 16:01 bin -> usr/bin/
       1 drwxr-xr-x   5 root root  2048 Dec 31  1969 boot/
       1 drwxr-xr-x  21 root root  4460 Nov 16 15:06 dev/
44302337 drwxr-xr-x  94 root root  4096 Nov 16 19:15 etc/
 3932161 drwxr-xr-x   3 root root  4096 Mar  2  2022 home/
      13 lrwxrwxrwx   1 root root     7 Oct 18 16:01 lib -> usr/lib/
      14 lrwxrwxrwx   1 root root     7 Oct 18 16:01 lib64 -> usr/lib/
      11 drwx------   2 root root 16384 Mar  2  2022 lost+found/
[...]
```

FILE API (ATTEMPT 2)

```
read(char *path, void *buf, off_t offset, size_t nbyte)
write(char *path, void *buf, off_t offset, size_t nbyte)
```

Disadvantages?

Expensive traversal!

Goal: Traverse once, not ever time we read or write

Three types of names:

1. ~~inode~~
2. ~~path~~
3. file descriptor

3) File Descriptor (FD)

Idea: Do expensive traversal once (open file)

- Store inode in descriptor object (kept in memory)
- Do reads/writes via descriptor, which tracks offset

Each process:

File-descriptor table contains pointers to open file descriptors

Integers used for file I/O are indexes into this per-process table

stdin: 0, stdout: 1, stderr: 2

FILE API (ATTEMPT 3)

```
int fd = open(char *path, int flag, mode_t mode)
read(int fd, void *buf, size_t nbyte)
write(int fd, void *buf, size_t nbyte)
close(int fd)
```

Advantages:

- human-readable names
- hierarchical
- traverse once
- offsets precisely defined

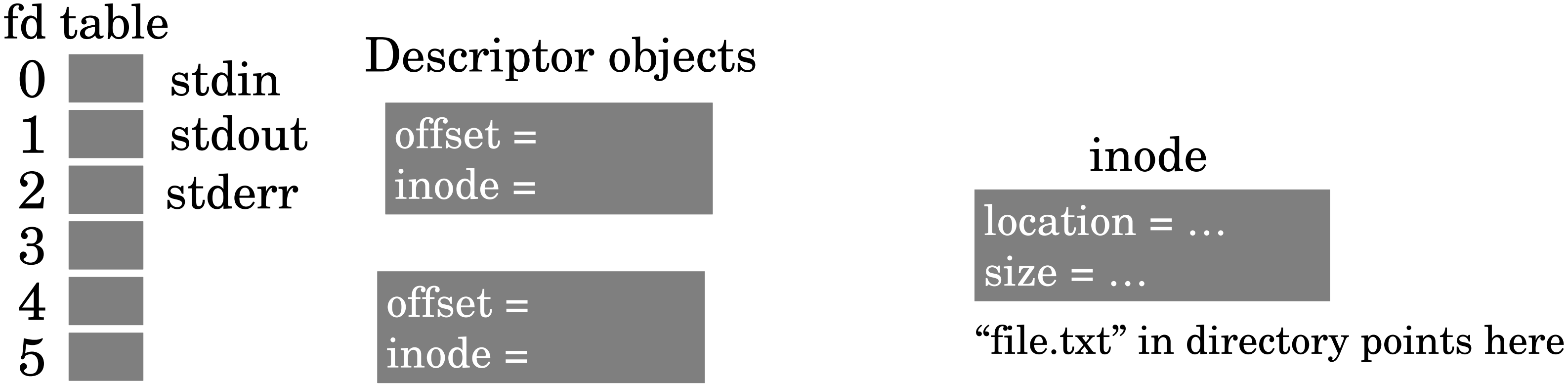
FD TABLE (XV6)

```
struct file {
  ...
  struct inode *ip;
  uint off;
};
```

```
// System-wide
struct {
    struct spinlock lock;
    struct file file[NFILE];
} ftable;
```

```
// Per-process state
struct proc {
  ...
  struct file *ofile[NOFILE];  // Open files
  ...
}
```

CODE SNIPPET: OPEN VS. DUP




```
int fd1 = open(“file.txt”); // returns 3
read(fd1, buf, 12);
int fd2 = open(“file.txt”); // returns 4
int fd3 = dup(fd2);         // returns 5
```

READ NON-SEQUENTIALLY

```
off_t lseek(int fildes, off_t offset, int whence)
```

- If whence is `SEEK_SET`, the offset is **set** to offset bytes
- If whence is `SEEK_CUR`, the offset is set to its current location **plus** offset bytes
- If whence is `SEEK_END`, the offset is set to the **size** of the file plus offset bytes

INTERACTION OF FILE DESCRIPTORS

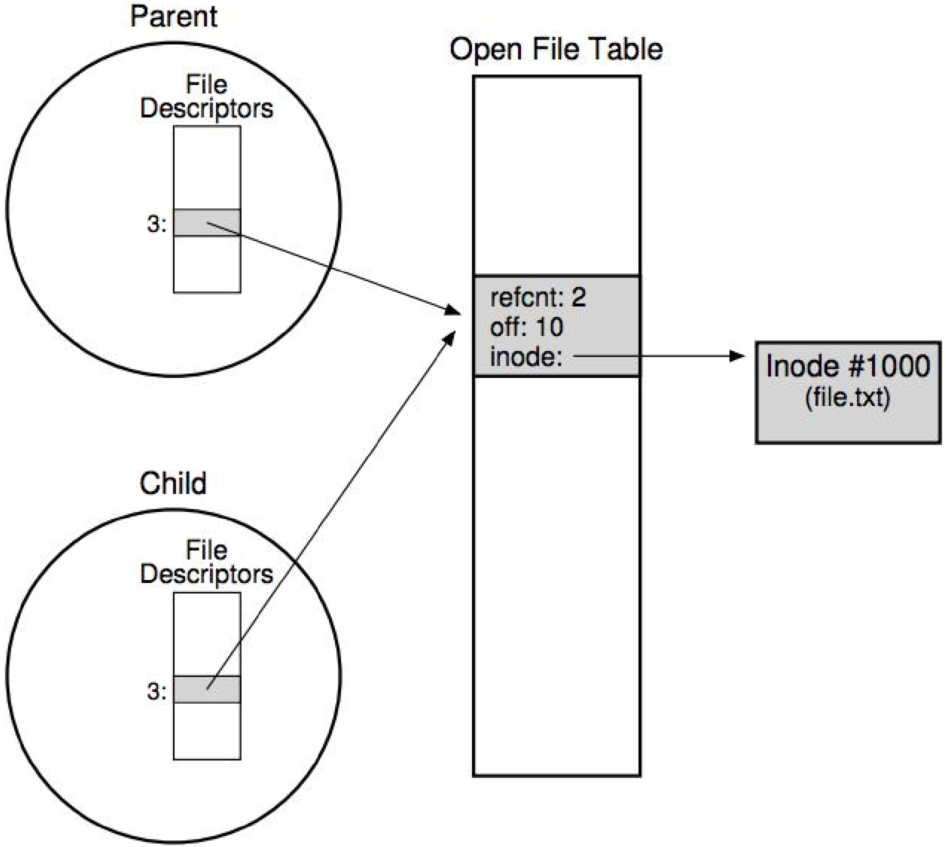
```
int fd1 = open("file.txt"); // returns 12
int fd2 = open("file.txt"); // returns 13
read(fd1, buf, 16);
int fd3 = dup(fd2);          // returns 14
read(fd2, buf, 16);
lseek(fd1, 100, SEEK_SET);
```

What are the value of offsets in fd1, fd2, fd3 after the above code sequence?

100, 16, 16

WHAT HAPPENS ON FORK()?

Man pages: The child process has its own copy of the parent's descriptors. These descriptors reference the same underlying objects, so that, for instance, file pointers in file objects are shared between the child and the parent, so that an lseek(2) on a descriptor in the child process can affect a subsequent read or write by the parent.



Deleting Files

There is no system call for deleting files!

Instead, calls to remove references (or different names) of those files

Inode (and associated file) is **freed** when there are no references

Two different types of references to inodes

- Path names are removed when `unlink()` is called
- FDs are removed when `close()` or process quits

LINKS: MULTIPLE NAMES FOR A FILE

Hard links: Both path names use same inode number

```
echo “Beginning…” > file1
ln file1 link
cat link
ls –li
echo “More info” >> file1
mv file1 file2
rm file2
```

Increment reference count in inode whenever add link

- File does not disappear until all removed (ref count = 0)

No differences across two files that are hard linked

- Links can be to files across directories
- Cannot hard link directories
  - Why not?

SOFT LINKS

Soft or symbolic links: Point to second path name

```
ln -s oldfile softlink
```

- Softlink will have new inode number
- Set bit in inode designating “soft link”; Interpret associated data as file name!
- Does not increment reference count
- It is possible to softlink to directories

How can you get confusing behavior: “file does not exist”!
Confusing behavior: “cd linked_dir; cd ..; in different parent!

LINKING: QUIZ

Consider the following code snippet:

```
echo “hello” > oldfile
ln –s oldfile link1
ln oldfile link2
rm oldfile
```

What will be the output of `cat link1`? `“No such file or directory”`

What will be the output of `cat link2`? `“hello”`

FILE SYNC

File system keeps newly written data in memory for awhile

- Buffer cache (portion of main memory, shared with virtual memory system)
- Useful for reads (don't have to access slow disk)

Also useful for writes

- Write buffering improves performance (why?)

But what if system crashes before buffers are flushed?

`fsync(int fd)` forces buffers to flush from memory to disk, tells disk to flush its write cache

- Makes data **durable**

What happens when you call `close(fd)`?

Man Pages for close(fd)

A successful close does not guarantee that the data has been successfully saved to disk, as thekernel uses the buffer cache to defer writes.

Typically, systemically do not flush buffers when a file is closed.

If you need to be sure that the data is physically stored on the underlying disk, use fsync(2).

(It will depend on the disk hardware at this point.)

RENAME

rename(char *old, char *new):
- deletes an old link to a file
- creates a new link to a file

Just changes name of file, does not move (or copy) data
Even when renaming to new directory

What can go wrong if system crashes at wrong time?

ATOMIC FILE UPDATE

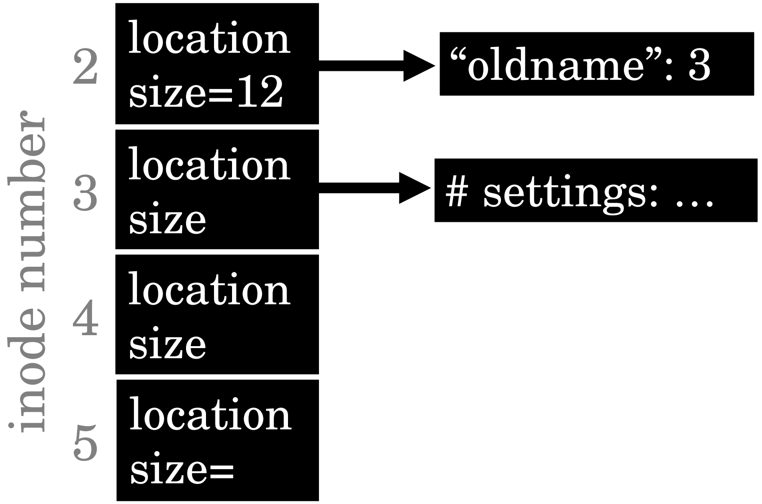
Rename operation must be atomic within file system

Common goal: Application wants to update file.txt atomically

- If crash, should see only old contents or only new contents

1. write new data to file.txt.tmp file
2. fsync file.txt.tmp
3. rename file.txt.tmp over file.txt, replacing it

Make operations atomic with journaling file systems



PERMISSIONS, ACCESS CONTROL

```
[yuvraj@zeus] (15)$ ls -lia
2117904663 drwxrwxr-x  4 yuvraj yuvraj  2048 Jul 19 18:45 ./
2040669394 drwxr-xr-x 64 yuvraj yuvraj 10240 Jul 19 18:44 ../
2117904665 drwxrwxr-x  2 yuvraj yuvraj  2048 Jul 19 18:44 dir1/
2117904681 drwxrwxr-x  2 yuvraj yuvraj  2048 Jul 19 18:44 dir2/
2118517764 -rw-rw-r--  1 yuvraj yuvraj     0 Jul 19 18:44 file1
2118517854 -rw-rw-r--  1 yuvraj yuvraj     0 Jul 19 18:44 file2
2118490104 lrwxr-xr-x  1 yuvraj yuvraj     5 Jul 19 18:45 sfile2 -> file2
```

```
[yuvraj@zeus] (16)$ fs la .
Access list for . is
Normal rights:
  system:administrators rlidwka
  system:anyuser l
  yuvraj rlidwka
```

MANY FILE SYSTEMS

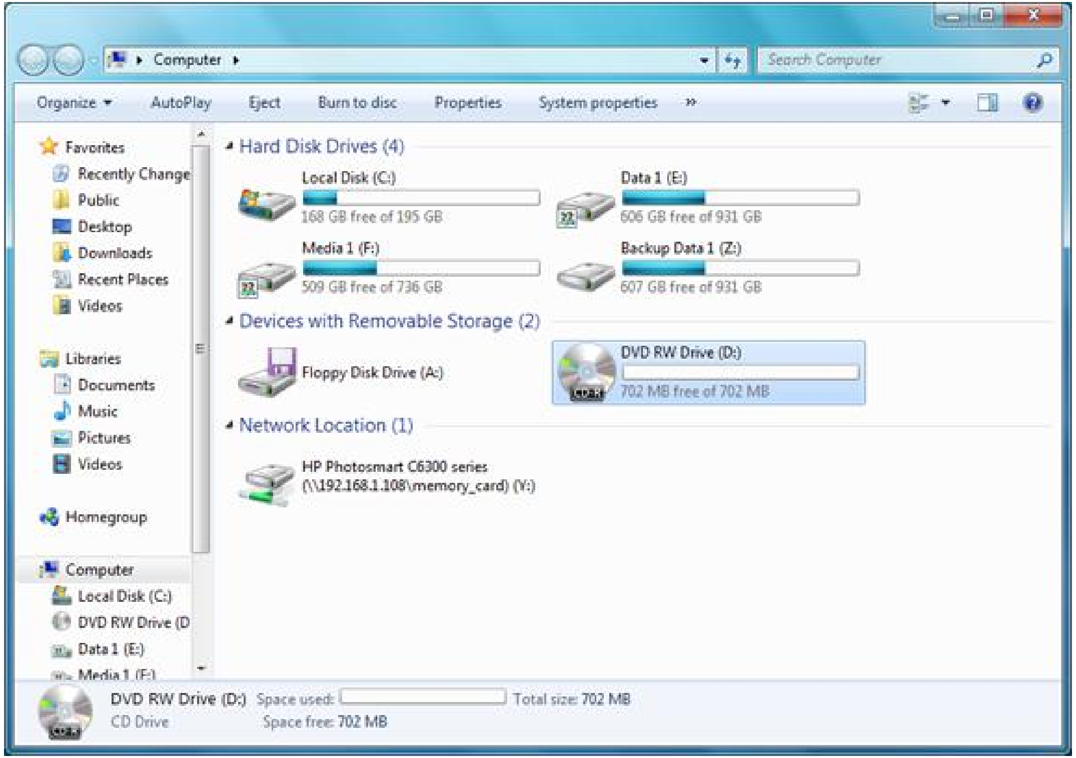
Users often want to use many file systems

For example:

- main disk/partition
- boot partition
- temporary files
- removable drives (CD-ROM, USB, etc.)

What is the most elegant way to support this?

MANY FILE SYSTEMS: APPROACH 1

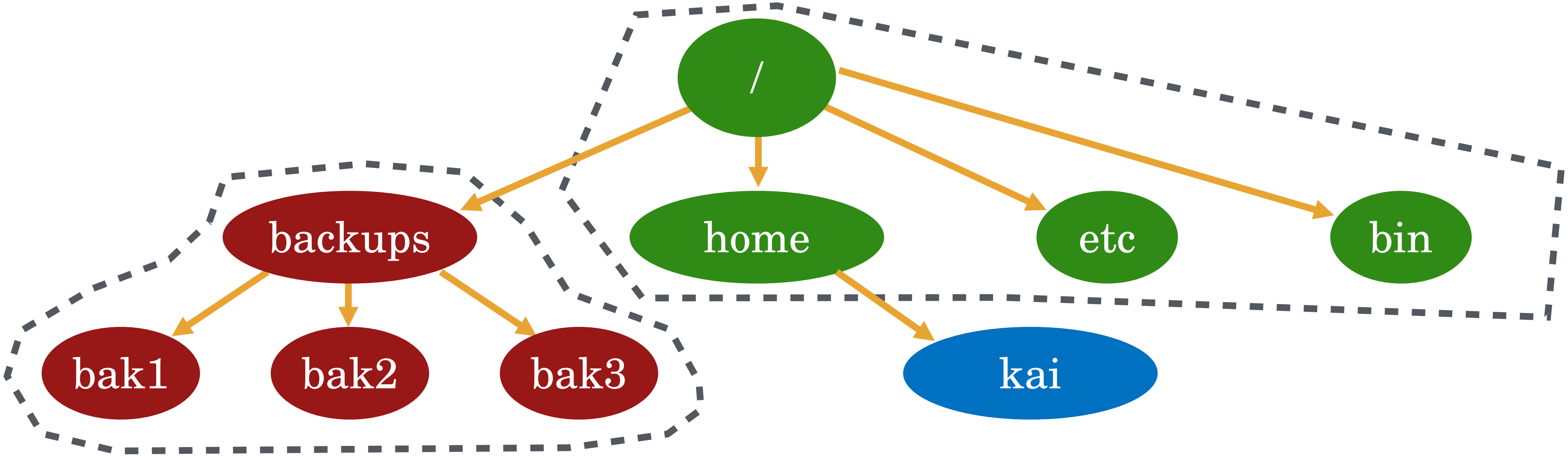


http://www.ofzenandcomputing.com/burn-files-cd-dvd-windows7/

MANY FILE SYSTEMS: APPROACH 2

Idea: stitch all the file systems together into a super file system!

```
sh> mount
/dev/sda1 on / type ext4 (rw)
/dev/sdb1 on /backups type ext4 (rw)
AFS on /home type afs (rw)
```



SUMMARY

Using multiple types of names provides convenience and efficiency

- inodes
- path names
- file descriptors

Special calls (`fsync`, `rename`) let developers communicate requirements to the file system

Mount and link features provide flexibility