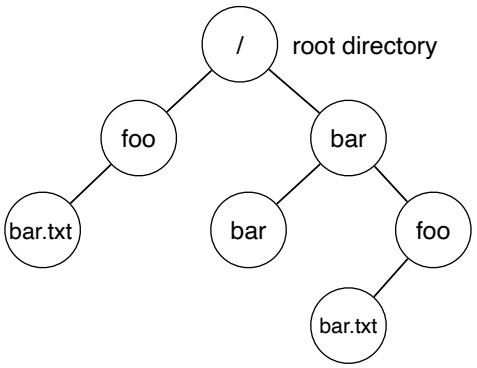
Persistence

- 36 - I/O Devices
- 37 - Hard Disk Drives
- 38 - RAID
- 39 - File and Directories
- 40 - File System Implementation
- 41 - Locality and the Fast File System
- 42 - Crash Consistency
- 43 - Log-structured File Systems
- 44 - Flash-based SSD
- 45 - Data Integrity and Protection

Persistence

- Keep data intact even if power loss
  - hard drive
  - solid-state device
- Two key abstractions
  - File
    - linear array of bytes
    - each has *inode number (inumber)*
  - Directory
    - stored like a file, has inumber
    - contains list of (user-readable name, low-level name) pairs
    - each entry refers to other file or directory.
- Assume a directory entry (“foo”, 10)
  - File “foo” w/ inode number (“inumber”) 10

Directory Hierarchy



Example Directory Tree

Valid files (absolute pathname) :
/foo/bar.txt
/bar/foo/bar.txt

Valid directory :
/
/foo
/bar
/bar/bar
/bar/foo/

Sub-directories

Creating Files

- Use `open()` system call with `O_CREAT` flag

```
int fd = open("foo", O_CREAT | O_WRONLY | O_TRUNC);
```

  - `O_CREAT` : create file
  - `O_WRONLY` : only write to that file while opened
  - `O_TRUNC` : truncate file (remove existing content)

- `open()` system call returns *file descriptor*
  - *File descriptor* is an integer, and is used to access files

Reading and Writing Files

- Reading and writing '`foo`':

```
prompt> echo hello > foo
prompt> cat foo
hello
prompt>
```

  - `echo` : redirect output to `foo`
  - `cat` : dump contents of file to screen

Reading and writing files…

```
prompt> strace cat foo
…
open("foo", O_RDONLY|O_LARGEFILE)  = 3
read(3, "hello\n", 4096)           = 6
write(1, "hello\n", 6)             = 6 // file descriptor 1: standard out
hello
read(3, "", 4096)                  = 0 // 0: no bytes left in the file
close(3)                           = 0
…
prompt>
```

- `open` (file descriptor, flags)
  - return file descriptor ("3" in example)
  - descriptors 0, 1, 2 are standard input, output, and error
- `read` (file descriptor, buffer pointer, buffer size)
  - returns number bytes read
- `write` (file descriptor, bufer pointer, buffer size)
  - return number of bytes written

Current Linux Host….

```
machine:~> strace cat hello
execve("/usr/bin/cat", ["cat", "hello"], 0x7ffe99db5388 /* 41 vars */) = 0
brk(NULL)                               = 0x64b0bb0a1000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7cc311a84000
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
fstat(3, {st_mode=S_IFREG|0644, st_size=57203, ...}) = 0
mmap(NULL, 57203, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7cc311a76000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/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\220\243\2\0\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\0"..., 784, 64) = 784
fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
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\0\0"..., 784, 64) = 784
mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7cc311800000
mmap(0x7cc311828000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) =
0x7cc311828000
mmap(0x7cc3119b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7cc3119b0000
mmap(0x7cc3119ff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) =
0x7cc3119ff000
mmap(0x7cc311a05000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7cc311a05000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7cc311a73000
arch_prctl(ARCH_SET_FS, 0x7cc311a73740) = 0
set_tid_address(0x7cc311a73a10)         = 190387
set_robust_list(0x7cc311a73a20, 24)     = 0
rseq(0x7cc311a74060, 0x20, 0, 0x53053053) = 0
mprotect(0x7cc3119ff000, 16384, PROT_READ) = 0
mprotect(0x64b07c440000, 4096, PROT_READ) = 0
mprotect(0x7cc311abc000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7cc311a76000, 57203)           = 0
getrandom("\x0e\xa0\xae\x32\xf8\x61\xfb\x14", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x64b0bb0a1000
brk(0x64b0bb0c2000)                     = 0x64b0bb0c2000
openat(AT_FDCWD, "/usr/lib/locale/locale-archive", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=3055776, ...}) = 0
mmap(NULL, 3055776, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7cc311400000
close(3)                                = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
openat(AT_FDCWD, "hello", O_RDONLY)     = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=6, ...}) = 0
fadvise64(3, 0, 0, POSIX_FADV_SEQUENTIAL) = 0
mmap(NULL, 139264, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7cc311a51000
read(3, "hello\n", 131072)              = 6
write(1, "hello\n", 6hello
)
                = 6
read(3, "", 131072)                     = 0
munmap(0x7cc311a51000, 139264)          = 0
close(3)                                = 0
close(1)                                = 0
close(2)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
machine:~>
```


Non-sequential Reading and Writing

- open files have *current offset*
  - specifies byte position where next read or write will begin
- updated
  - implicitly: a read of `N` bytes adds `N` to current offset
  - explicitly: `lseek()`

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

- `fildes` : file descriptor
- `offset` : location in file
- whence : how the seek is performed

From the man page:

```
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.
```

fsync()

- file system *buffers* writes in memory
  - overwrites
  - write combining
  - push
  - *data can be lost*
- some apps require more timely guarantees
  - DBs….
  - `fsync()` returns once the data has made it's way to the disk

```
int fd = open("foo", O_CREAT | O_WRONLY | O_TRUNC);
assert (fd > -1)
int rc = write(fd, buffer, size);
assert (rc == size);
rc = fsync(fd);
assert (rc == 0);
```


Other FS APIs

- `rename (char *old, char *new)` (*atomic*)

```
prompt> mv foo bar        // mv uses the system call rename()
```

- `often used by editors to write new versions`

```
int fint fd = open("foo.txt.tmp", O_WRONLY|O_CREAT|O_TRUNC);
write(fd, buffer, size); // write out new version of file
fsync(fd);
close(fd);
rename("foo.txt.tmp", "foo.txt");
```

Other FS APIs

- `stat()`, `fstat()` : show metadata

```
struct stat {
    dev_t st_dev;         /* ID of device containing file */
    ino_t st_ino;         /* inode number */
    mode_t st_mode;       /* protection */
    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 filesystem I/O */
    blkcnt_t st_blocks;   /* number of 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 */
};
```


Stat, cont.

```
prompt> echo hello > file
prompt> stat file

File: 'file'
Size: 6 Blocks: 8 IO Block: 4096 regular file
Device: 811h/2065d Inode: 67158084 Links: 1
Access: (0640/-rw-r-----) Uid: (30686/ root) Gid: (30686/ remzi)
Access: 2011-05-03 15:50:20.157594748 -0500
Modify: 2011-05-03 15:50:20.157594748 -0500
Change: 2011-05-03 15:50:20.157594748 -0500
```

- info kept in `inode`

Removing files

- `rm` calls `unlink()`

```
prompt> strace rm foo
…
unlink("foo")              = 0      // return 0 upon success
…
prompt>
```

- `link(old *char, *new)    // "hard" link`
  - adds a new hard path to get to same file

- inode tracks number of links
  - *deletes inode when zero*

Symbolic ("soft") links

- implemented as third type of file/dir
  - file contains a static path to redirect accesses
  - can link directories (hard links cannot)
  - inode is not updated, so symbolic links can be *orphaned*

Links cont.

```
hyperion:~/x> echo "hello" > file
hyperion:~/x> ln -s file file2
hyperion:~/x> ln file file3
hyperion:~/x> ll
total 16K
drwxr-xr-x  2 keleher keleher 4.0K Mar 26 10:24 .
drwxr-xr-x 23 keleher keleher 4.0K Mar 26 10:23 ..
-rw-r--r--  2 keleher keleher    6 Mar 26 10:23 file
lrwxrwxrwx  1 keleher keleher    4 Mar 26 10:24 file2 -> file
-rw-r--r--  2 keleher keleher    6 Mar 26 10:23 file3
hyperion:~/x> stat file
  File: file
  Size: 6          Blocks: 8          IO Block: 4096   regular file
Device: 802h/2050d Inode: 159776897   Links: 2
Access: (0644/-rw-r--r--)  Uid: ( 1000/ keleher)   Gid: ( 1000/ keleher)
Access: 2024-03-26 10:23:57.714939940 -0400
Modify: 2024-03-26 10:23:57.714939940 -0400
Change: 2024-03-26 10:24:19.990594144 -0400
 Birth: 2024-03-26 10:23:57.714939940 -0400
hyperion:~/x> mv file nfile
hyperion:~/x> cat nfile
hello
hyperion:~/x> cat file2
cat: file2: No such file or directory
hyperion:~/x> cat file3
hello
hyperion:~/x>
```


Making and mounting file systems

- `mkfs [-t <type>] <device> [<size>]`
  - make empty file system
  - creates a root directory on given partition

- `mount [-t <type>] <device> <dir>`
  - maps file system on top of an existing directory

```
prompt> mount -t ext3 /dev/sda1 /home/users
prompt> ls /home/users
a b
```

Linux machine

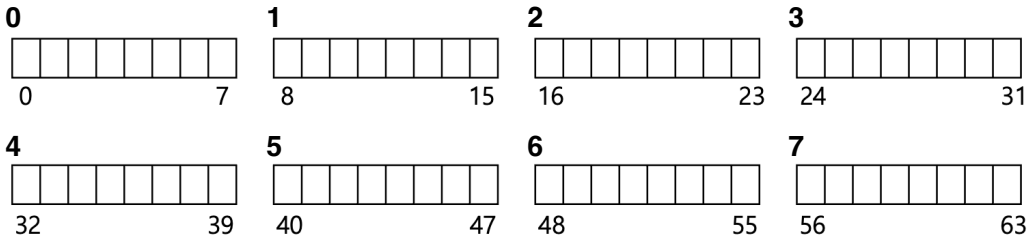
```
hyperion:~> mount
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs (rw,nosuid,relatime,size=8087356k,nr_inodes=2021839,mode=755,inode64)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)
tmpfs on /run type tmpfs (rw,nosuid,nodev,noexec,relatime,size=1629064k,mode=755,inode64)
/dev/sda2 on / type ext4 (rw,relatime,errors=remount-ro)
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev,inode64)
tmpfs on /run/lock type tmpfs (rw,nosuid,nodev,noexec,relatime,size=5120k,inode64)
cgroup2 on /sys/fs/cgroup type cgroup2 (rw,nosuid,nodev,noexec,relatime)
pstore on /sys/fs/pstore type pstore (rw,nosuid,nodev,noexec,relatime)
bpf on /sys/fs/bpf type bpf (rw,nosuid,nodev,noexec,relatime,mode=700)
systemd-1 on /proc/sys/fs/binfmt_misc type autofs (rw,relatime,fd=29,pgrp=1,timeout=0,minproto=5,maxproto=5,direct,pipe_ino=21621
mqueue on /dev/mqueue type mqueue (rw,nosuid,nodev,noexec,relatime)
debugfs on /sys/kernel/debug type debugfs (rw,nosuid,nodev,noexec,relatime)
hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime,pagesize=2M)
tracefs on /sys/kernel/tracing type tracefs (rw,nosuid,nodev,noexec,relatime)
fusectl on /sys/fs/fuse/connections type fusectl (rw,nosuid,nodev,noexec,relatime)
configfs on /sys/kernel/config type configfs (rw,nosuid,nodev,noexec,relatime)
none on /run/credentials/systemd-sysusers.service type ramfs (ro,nosuid,nodev,noexec,relatime,mode=700)
tracefs on /sys/kernel/debug/tracing type tracefs (rw,nosuid,nodev,noexec,relatime)
/var/lib/snapd/snaps/core20_2182.snap on /snap/core20/2182 type squashfs (ro,nodev,relatime,errors=continue,x-gdu.hide)
/var/lib/snapd/snaps/snapd_20671.snap on /snap/snapd/20671 type squashfs (ro,nodev,relatime,errors=continue,x-gdu.hide)
/var/lib/snapd/snaps/lxd_24061.snap on /snap/lxd/24061 type squashfs (ro,nodev,relatime,errors=continue,x-gdu.hide)
/var/lib/snapd/snaps/core20_2105.snap on /snap/core20/2105 type squashfs (ro,nodev,relatime,errors=continue,x-gdu.hide)
/dev/sdb1 on /media/littlessd type ext4 (rw,relatime,stripe=256)
/dev/sdc1 on /media/bigssd type ext4 (rw,relatime,stripe=512)
binfmt_misc on /proc/sys/fs/binfmt_misc type binfmt_misc (rw,nosuid,nodev,noexec,relatime)
lxcfs on /var/lib/lxcfs type fuse.lxcfs (rw,nosuid,nodev,relatime,user_id=0,group_id=0,allow_other)
tmpfs on /run/snapd/ns type tmpfs (rw,nosuid,nodev,noexec,relatime,size=1629064k,mode=755,inode64)
nsfs on /run/snapd/ns/lxd.mnt type nsfs (rw)
/var/lib/snapd/snaps/snapd_21184.snap on /snap/snapd/21184 type squashfs (ro,nodev,relatime,errors=continue,x-gdu.hide)
tmpfs on /run/user/1000 type tmpfs (rw,nosuid,nodev,relatime,size=1629060k,nr_inodes=407265,mode=700,uid=1000,gid=1000,inode64)
hyperion:~>
```


Persistence

- 36 - I/O Devices
- 37 - Hard Disk Drives
- 38 - RAID
- 39 - File and Directories
- 40 - File System Implementation
- 41 - Locality and the Fast File System
- 42 - Crash Consistency
- 43 - Log-structured File Systems
- 44 - Flash-based SSD
- 45 - Data Integrity and Protection

Let's Start With Blocks…

- File systems address disks by *block*
  - *Logical* block numbers are an arbitrary mapping over physical
  - blocks are multiples of disk *sectors*
  - usually 8k or 16k (512 bytes for GeekOS)
- Assume 512-byte sectors and 4k pages in the following
  - physical block numbers start at 0:

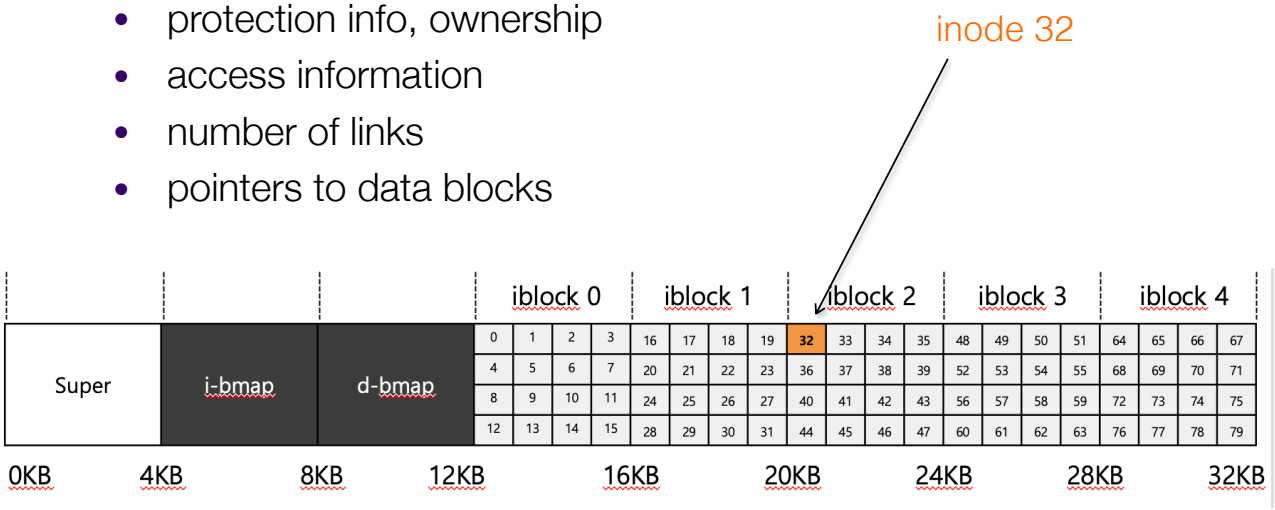


Disk Organization

- Blocks on disk
  - *super block*
    - configuration for a specific file system instance
    - boot code
    - size and location of inode tables, etc.
  - *data blocks*
    - blocks containing file data
  - *inode blocks*
    - inode structures w/ file metadata
  - *indirect pointer blocks*
    - blocks full of pointers to other blocks
  - *bitmaps*
    - used/free information for data and inode blocks

Simplified Disk Layout

- array of per-file metadata in *inodes:*
  - *inumber* : index into the inode table
  - file type (regular file, directory, etc.)
  - size, number of blocks
  - protection info, ownership
  - access information
  - number of links
  - pointers to data blocks



Ext2 (old linux) Inodes

Size	Name	What is this inode field for?
2	mode	can this file be read/written/executed?
2	uid	who owns this file?
4	size	how many bytes are in this file?
4	time	what time was this file last accessed?
4	ctime	what time was this file created?
4	mtime	what time was this file last modified?
4	dtime	what time was this inode deleted?
4	gid	which group does this file belong to?
2	links_count	how many hard links are there to this file?
2	blocks	how many blocks have been allocated to this file?
4	flags	how should ext2 use this inode?
4	osd1	an OS-dependent field
60	block	**a set of disk pointers** (15 total), often 2 indrect, 1 doubly indirect
4	generation	file version (used by NFS)
4	file_acl	a new permissions model beyond mode bits
4	dir_acl	called access control lists
4	faddr	an unsupported field
12	i_osd2	another OS-dependent field

Indirect blocks

- W/ 512-bytes blocks, 4-byte pointers:
  - the 15 block pointers can accommodate files up to size 7.5K
- W/ a single level of indirection:
  - $15 * \frac{512}{4} \times 512 = 15 \times 2^{16} = 983\text{K}$, much larger!
- W/ double indirection:
  - $15 \times \frac{512}{4} \times \frac{512}{4} \times 512 = 15 \times 2^{23} = 120\text{MB}$
- Used in most large file systems:
  - Linux EXT2, EXT3, NetApp's WAFL, Unix file system
  - Linux EXT4 uses *extents* instead of simple pointers
    - extent lets a pointer reference consecutive blocks

File System Numbers

• Rules of thumb:	
• most are small	*2K most common*
• avg size is growing	*over 200K*
• most bytes are in large files	
• there are many	*100K on average*
• most FS are about half full	*disk size grows, so do files*
• directories typically small	*most have 20 or fewer*

Test #2

- Concurrency
- Posix theads API (not the syntax)
- Overview, and POSIX threads (pthreads)
- Primitives:
  - Locks
  - Condition Variables
  - Semaphores
- Concurrent Data Structures
- Common issues:
  - deadlocks, livelocks, starvation, fairness
- Classic problems:
  - baboons, producer-consumer, dining philosophers
- Event-Based Concurrency

Test #2

- (20 pts) short answer
- (10 pts) identifying problem in code
- (20 pts) using and talking about atomic instructions
- (10 pts) safe, unsafe, and deadlocked allocation graphs
- (30 pts) writing semaphore code
- (10 pts) identifying and using wait-for graph