Lecture Note 6. File System Basic

April 28, 2020

Jongmoo Choi

Dept. of software
Dankook University
http://embedded.dankook.ac.kr/~choijm

Contents

Chap. 35 A Dialogue on Persistence

Professor: *And thus we reach the third of our four ... err... three pillars of operating systems:* ***persistence****.*

Student: *Did you say there were three pillars, or four? What is the fourth?*

Professor: *No. Just three, young student, just three. Trying to keep it simple here.*

Student: *OK, fine. But what is persistence, oh fine and noble professor?*

Professor: *Actually, you probably know what it means in the traditional sense, right? As the dictionary would say: "a firm or obstinate continuance in a course of action in spite of difficulty or opposition."*

Student: *It's kind of like taking your class: some obstinance required.*

Professor: *Ha! Yes. But persistence here means something else. Let me explain. Imagine you are outside, in a field, and you pick a —*

Student: *(interrupting) I know! A peach! From a peach tree!*

Professor: *I was going to say apple, from an apple tree. Oh well; we'll do it your way, I guess.*

Student: *(stares blankly)*

Professor: *Anyhow, you pick a peach; in fact, you pick many many peaches, but you want to make them last for a long time. Winter is hard and cruel in Wisconsin, after all. What do you do?*

Student: *Well, I think there are some different things you can do. You can pickle it! Or bake a pie. Or make a jam of some kind. Lots of fun!*

Professor: *Fun? Well, maybe. Certainly, you have to do a lot more work to make the peach* ***persist****. And so it is with information as well; making information persist, despite computer crashes, disk failures, or power outages is a tough and interesting challenge.*

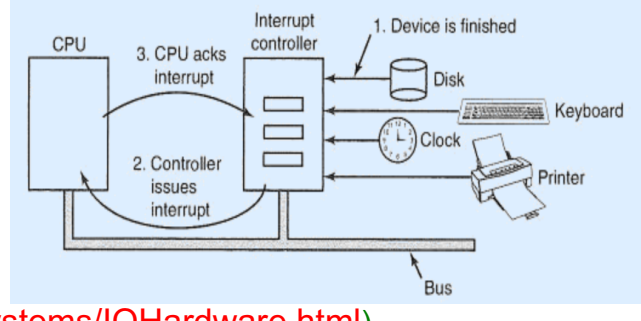
Student: *Nice segue; you're getting quite good at that.*

Professor: *Thanks! A professor can always use a few kind words, you know.*

? Persistence : Making information durable despite of computer crash, disk failures and so on

Chap. 36 I/O Devices

36.1 System Architecture
36.2 A Canonical Device
36.3 The Canonical Protocol
36.4 Lowering CPU overhead with Interrupt
36.5 More Efficient Data Movement with DMA
36.6 Methods of Device Interaction
36.7 Fitting into the OS: The Device Driver
36.8 Case Study: A simple IDE Disk Driver
36.9 Historical Notes



(Source: https://gcallah.github.io/OperatingSystems/IOHardware.html)

36.1 System Architecture

Computer system focusing on Bus

- ✓ Hierarchical structure
  - Memory bus (System bus): CPU and Memory
    - Fast, Expensive, Short
  - I/O bus: SCSI, SATA, USB (and/or separated bus for Graphic Cards)
    - Slow, Less expensive, long, pluggable
- ✓ Modern system
  - Special interconnect: Memory interconnect (e.g. QPI, Hyperport), Graphic interconnect
  - Make use of specialized chipsets: I/O chips with different interfaces

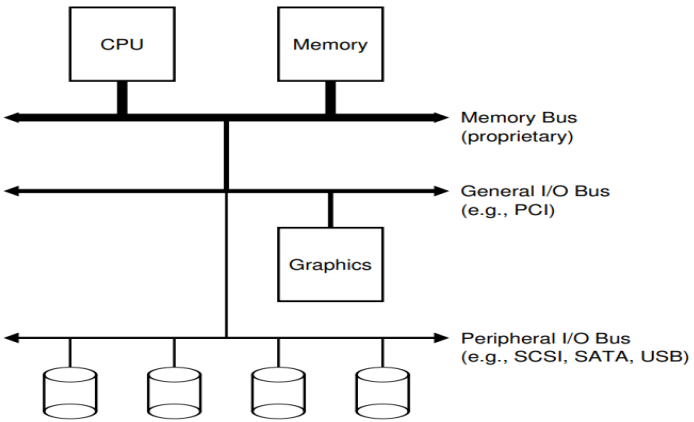


Figure 36.1: **Prototypical System Architecture**

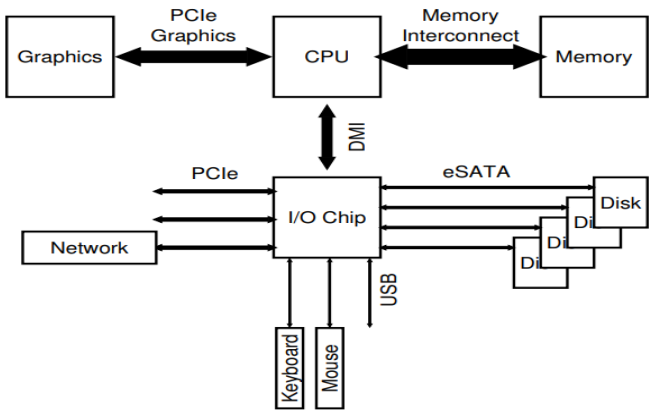


Figure 36.2: **Modern System Architecture**

36.2 A Canonical Device / 36.3 The Canonical Protocol

Devices

- ✓ Interface parts
  - Registers: command, status, data
- ✓ Internals
  - Logic: controller and special chips (device specific) + SW (called firmware)
  - Memory: I/O Buffer (e.g. store receiving packet, delayed write, …)

Protocol

- ✓ How to interact with devices?
  - Example: Four steps 1) idle check, 2) data, 3) command, 4) finish check
- ✓ 3 mechanisms: PIO(Programmed I/O), Interrupt, DMA
  - PIO: CPU performs all steps including idle/finish checking (polling)

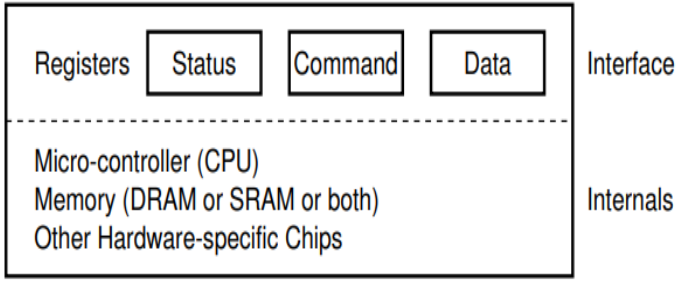


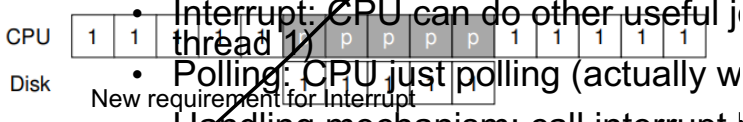
Figure 36.3: **A Canonical Device**

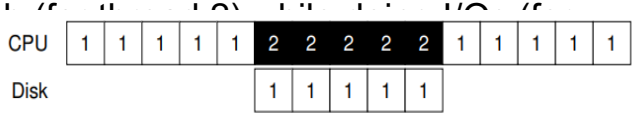
```
While (STATUS == BUSY)
    ; // wait until device is not busy
Write data to DATA register
Write command to COMMAND register
    (starts the device and executes the command)
While (STATUS == BUSY)
    ; // wait until device is done with your request
```

36.4 Lowering CPU overhead with Interrupt

Interrupt vs Polling

- ✓ Comparison
  - Polling: Checking status (busy or idle, like spin) ? thread state: running (still hold CPU while its usage is only checking device status)
  - Interrupt: Inform when device is idle (or work is done) ? thread state: sleeping (release CPU which can be utilized usefully by other threads
    - Note) Interrupt definition: a mechanism that informs an event to OS
- ✓ Example
  - Thread 1 requests disk access (read or write)



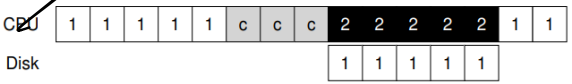


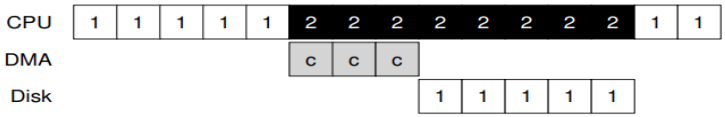
- ✓ Tradeoffs
  - Benefit of Interrupt: overlapping
    - Interrupt: CPU can do other useful j
    - thread 1)
    - Polling: CPU just polling (actually w
  - New requirement for Interrupt
    - Handling mechanism: call interrupt handler via interrupt table (page 28 in LN 2)
    - Sleep queue management (Context switch overhead)
  - Usage suggestion (depend on devices)
    - Slow device: Interrupt, Fast device: Polling (like spin and sleep lock)
    - Optimization: Hybrid, Interrupt coalescing

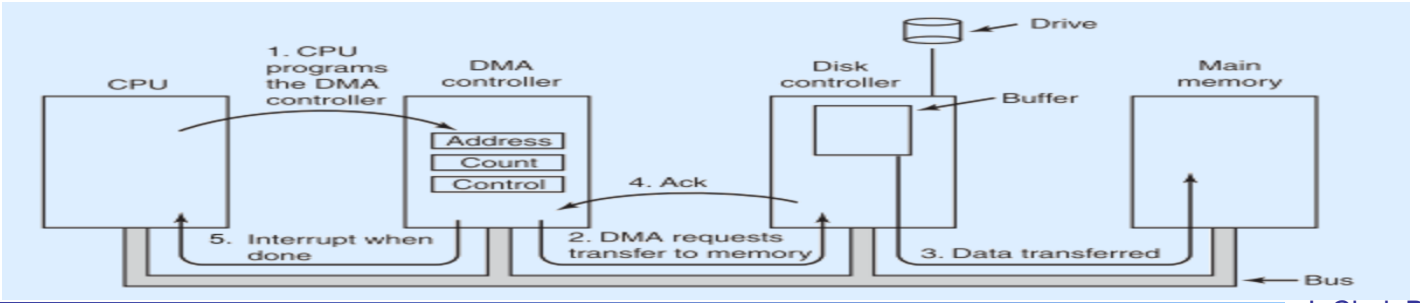
36.5 More Efficient Data Movement with DMA

DMA (Direct Memory Access)

- ✓ Comparison
  - PIO (Programmed I/O): CPU manages data copy between memory and devices
    - Concern: Devices are too slow for CPU (note CPU: ns, Disk: ms)
  - DMA controller performs data copy between memory and devices
    - CPU can do other useful job (better overlapping)
- ✓ Example
  - Thread 1 requests disk write without/with DMA using Interrupt
  - Data copy (denoted as “c” in the figure is done by CPU vs. DMA)
- ✓ DMA mechanism



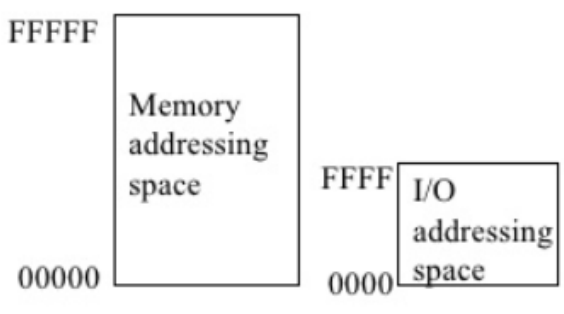




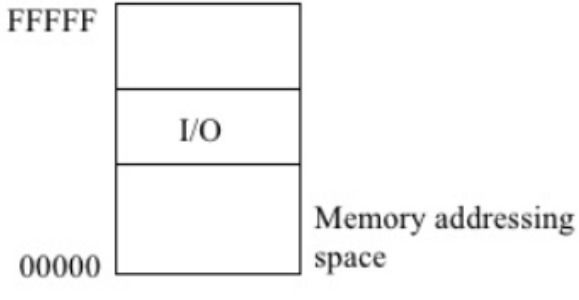
36.6. Methods of Device Interaction

How to address registers in devices?

- ✓ Two approaches
  - Direct I/O
    - Separated address space
    - Explicit I/O instruction (e.g. in/out + port)
  - Memory-mapped I/O
    - Single address space: DRAM + I/Os
    - Memory access instruction (e.g. load/store + I/O address space)
- ✓ Privileged instruction
  - Kernel mode: okay vs User mode: protection fault
  - Usually accessed in a kernel component called device driver



Direct I/O



Memory-mapped I/O

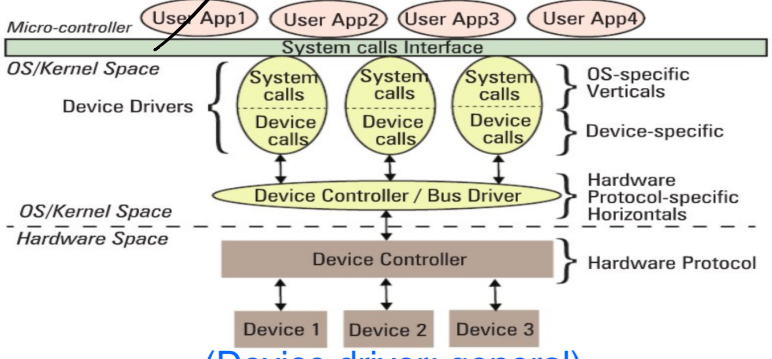
36.7 Fitting into the OS: The Device Driver

Device driver

- ✓ A set of software in kernel that abstracts devices
- ✓ Two layers
  - Manage 1) device registers (command, status, data), 2) interrupt and 3) DMA
  - Support generic interface such as open, read, write, close, … (like file)
- ✓ 70% of codes in Linux is device drivers (mostly kernel module)

Layered architecture

- ✓ Character device (or raw mode): device accessed by user directly
  - System call ? Driver ? Devices
- ✓ Block device: device accessed by user through file system (FS)
  - System call ? FS ? Block layer (buffer, scheduler) ? Driver ? Devices



(Device driver: general)

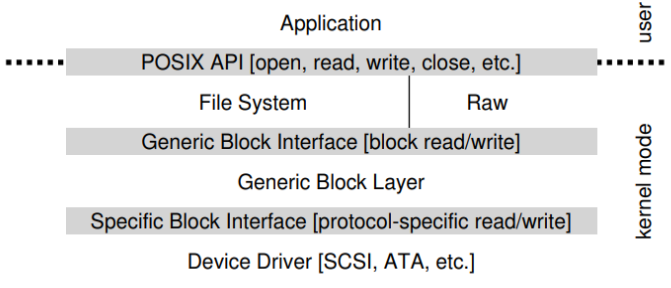


Figure 36.4: **The File System Stack**

(Device driver: block driver specific)

36.8 Case Study: A simple IDE Disk Driver (Optional)

A simple IDE disk controller

- ✓ Direct I/O (separated I/O address), I/O instruction: in/out
- ✓ 4 Registers

Control (0x3f6), Command block (0x1f2~1f6), Command or Status (0x1f7), Data port (0x1f0), Error (0x1f1)

- Note) 1) LBA: Logical Block Address, 2) Status: Busy/Ready, 3) Error: bad block,…

Example (low-level interface)

- Wait for drive ready: read 0x1f7 until the READY bit is on
- Write: Write sector count and LBA in 0x1f2~1f6 and Start I/O by writing WRITE command in 0x1f7
- Data transfer: wait until READY and DRQ (Drive Request for Data), write data into the Data port

```
Control Register:
   Address 0x3F6 = 0x08 (0000 1RE0): R=reset,
                   E=0 means "enable interrupt"

Command Block Registers:
   Address 0x1F0 = Data Port
   Address 0x1F1 = Error
   Address 0x1F2 = Sector Count
   Address 0x1F3 = LBA low byte
   Address 0x1F4 = LBA mid byte
   Address 0x1F5 = LBA hi  byte
   Address 0x1F6 = 1B1D TOP4LBA: B=LBA, D=drive
   Address 0x1F7 = Command/status

Status Register (Address 0x1F7):
     7       6      5      4      3      2      1      0
   BUSY   READY  FAULT  SEEK   DRQ   CORR  IDDEX  ERROR

Error Register (Address 0x1F1): (check when ERROR==1)
     7       6      5      4      3      2      1      0
   BBK     UNC     MC    IDNF   MCR   ABRT  TONF  AMNF

   BBK   = Bad Block
   UNC   = Uncorrectable data error
   MC    = Media Changed
   IDNF  = ID mark Not Found
   MCR   = Media Change Requested
   ABRT  = Command aborted
   TONF  = Track 0 Not Found
   AMNF  = Address Mark Not Found
```

Figure 36.5: The IDE Interface

36.8 Case Study: A simple IDE Disk Driver (Optional)

Driver interface (OS-level interface)

- ✓ Character driver: open, read, write, close, intr, …
- ✓ Block driver: open, close, intr, rw (or request, strategy), …

Note: dynamic loadable kernel module interface for Linux (insmod, rmmod)

IDE disk driver example: 4 main functions

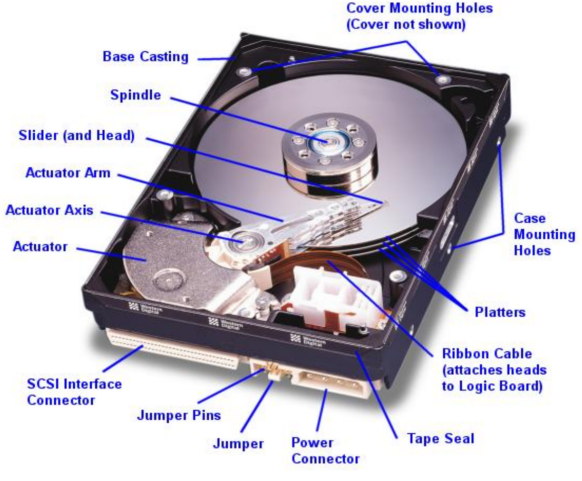
- ✓ ide_rw() ? ide_wait_ready() ? ide_start_request()
- ✓ ide_intr()

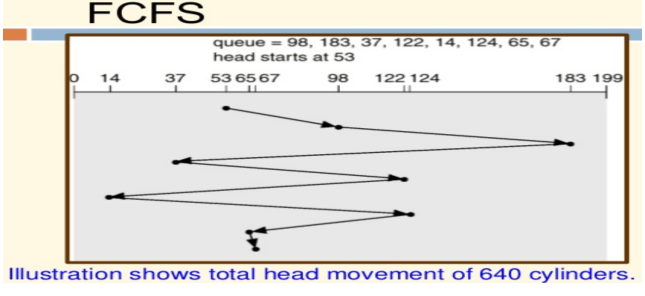
```
static int ide_wait_ready() {
  while (((int r = inb(0x1f7)) & IDE_BSY) || !(r & IDE_DRDY))
    ;  // loop until drive isn't busy
}

static void ide_start_request(struct buf *b) {
  ide_wait_ready();
  outb(0x3f6,  0);                // generate interrupt
  outb(0x1f2,  1);                // how many sectors?
  outb(0x1f3,  b->sector & 0xff);  // LBA goes here ...
  outb(0x1f4,  (b->sector >> 8) & 0xff);    // ... and here
  outb(0x1f5,  (b->sector >> 16) & 0xff);   // ... and here!
  outb(0x1f6,  0xe0 | ((b->dev&1)<<4) | ((b->sector>>24)&0x0f));
  if(b->flags & B_DIRTY){
    outb(0x1f7, IDE_CMD_WRITE);       // this is a WRITE
    outsl(0x1f0, b->data, 512/4);     // transfer data too!
  } else {
    outb(0x1f7, IDE_CMD_READ);        // this is a READ (no data)
  }
}

void ide_rw(struct buf *b) {
  acquire(&ide_lock);
  for (struct buf **pp = &ide_queue; *pp; pp=&(*pp)->qnext)
    ;                               // walk queue
  *pp = b;                          // add request to end
  if (ide_queue == b)               // if q is empty
    ide_start_request(b);           // send req to disk
  while ((b->flags & (B_VALID|B_DIRTY)) != B_VALID)
    sleep(b, &ide_lock);            // wait for completion
  release(&ide_lock);
}

void ide_intr() {
  struct buf *b;
  acquire(&ide_lock);
  if (!(b->flags & B_DIRTY) && ide_wait_ready() >= 0)
    insl(0x1f0, b->data, 512/4);    // if READ: get data
  b->flags |= B_VALID;
  b->flags &= ~B_DIRTY;
  wakeup(b);                        // wake waiting process
  if ((ide_queue = b->qnext) != 0)  // start next request
    ide_start_request(ide_queue);   // (if one exists)
  release(&ide_lock);
}
```

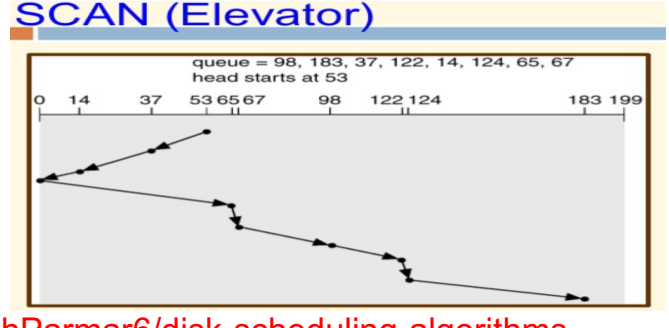
Figure 36.6: **The xv6 IDE Disk Driver (Simplified)**

37 Hard Disk Drives

37.1 The interface
37.2 Basic Geometry
37.3 A Simple Disk Drive
37.4 I/O Time: Doing the Math
37.5 Disk Scheduling







(Source: https://www.slideshare.net/PareshParmar6/disk-scheduling-algorithms-71247712)

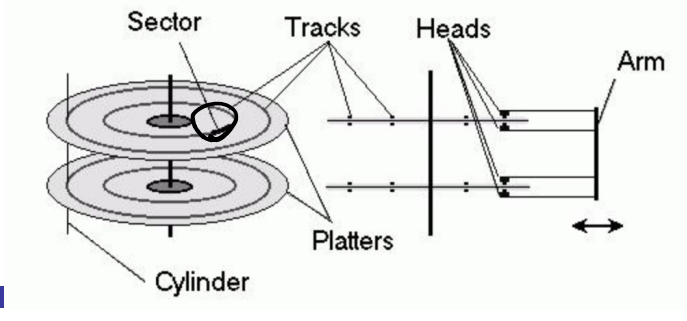
37.1 The interface / 37.2 Basic Geometry

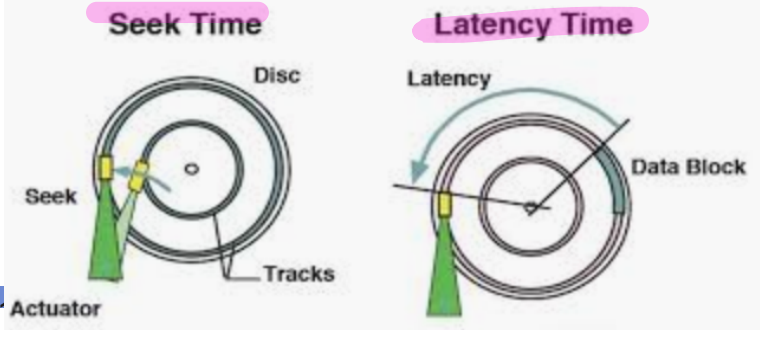
Interface

- ✓ Basic unit: sectors (512-byte)
  - Disk consists of a large number of sectors (0 ~ N-1 sectors or address space)
- ✓ Addressing (LBA: logical block address for disk)
  - Sector addressing: 512B
  - Multi-sector addressing (usually called as a disk block): 4KB or 8KB ? Kernel developer's viewpoint: disk is a set of disk blocks whose size is 4KB

Basic Geometry

- ✓ Platter (two surface) ? Track (thousands tracks per surface) ? Sectors
- ✓ Head: sensing data
  - Multiple heads (one per each surface), connected into an arm
- Data access: seek time + rotation time (latency) + transfer time
  - Cylinder: a set of same tracks in each surface (no seek time required)





37.3 A simple Disk Drive

In a same track access: Figure 37.2

- ✓ Assume
  - 12 sectors in a track, original head position is 6, target is 10
  - 10,000 RPM (rotation per minute) ? 1/6 rotation per ms (millisecond) ? a rotation takes 6ms
  - Rotational latency ? 2ms in this case (3ms on average)

Multiple tracks: Figure 37.3

- ✓ Original head position is 30, target is 11
- ✓ Need not only rotational latency but also seek time (ms)
  - Note that seek and rotational latency perform in parallel

Track skew: Figure 37.4

- ✓ To optimize sequential access (e.g. read sector 10, 11, 12, 13)
- ✓ Other optimizations: multi-zones, disk cache (track buffer)

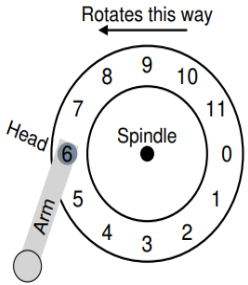
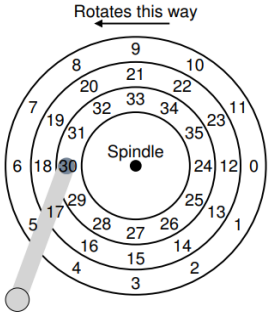


Figure 37.2: **A Single Track Plus A Head**



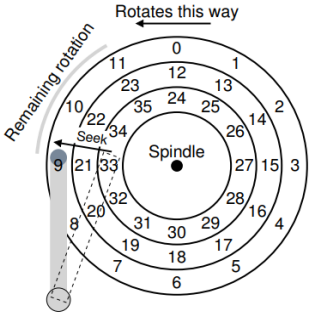


Figure 37.3: **Three Tracks Plus A Head (Right: With Seek)**

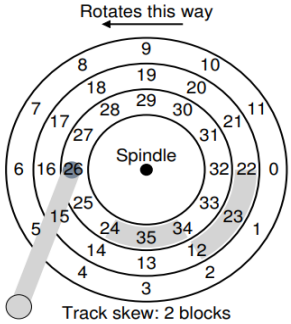


Figure 37.4: **Three Tracks: Track Skew Of 2**

37.4 I/O Time: Doing the Math

Metrics
- ✓ I/O time (latency)
- ✓ I/O rate (bandwidth)

$$T_{I/O} = T_{seek} + T_{rotation} + T_{transfer}$$

$$R_{I/O} = \frac{Size_{Transfer}}{T_{I/O}}$$

Workload
- ✓ Random: issues small (e.g., 4KB) reads to random locations on disk
- ✓ Sequential: reads a large number of sectors consecutively (100 MB)

Disk considered: Figure 37.5

- Simple question to take attendance: In 16 page, we compare IO rate between Cheetah and Barracuda. Calculate all components time (Tseek, Trotation, Ttransfer) for Cheetah and Barracuda and prove the correctness of values shown in Figure 37.6 (hint: refer to 7~8 page of chapter 37 in OSTEP). (until 6 PM, May 13th)
  - ✓ Random: Seek + rotation + transfer per 4KB
  - ✓ Sequential: One seek/rotation per large data (e.g. 100MB)

Sequential is much fast than random in disk
SW engineers need to make programs that access disks in sequential

	Cheetah 15K.5	Barracuda
Capacity	300 GB	1 TB
RPM	15,000	7,200
Average Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s
Platters	4	4
Cache	16 MB	16/32 MB
Connects via	SCSI	SATA

Figure 37.5: **Disk Drive Specs: SCSI Versus SATA**

	Cheetah	Barracuda
$R_{I/O}$ Random	0.66 MB/s	0.31 MB/s
$R_{I/O}$ Sequential	125 MB/s	105 MB/s

Figure 37.6: **Disk Drive Performance: SCSI Versus SATA**

37.5 Disk Scheduling

Disk scheduler

- ✓ Role: Examines I/O requests and decides which one to schedule next

Examples

- ✓ FCFS (First Come First Serve)
  Pros) simple, Cons) may cause long seek distance
- ✓ SSTF (Shortest Seek Time First)
  Pros) reduce seek distance, Cons) unfair (especially boundary tracks)
- ✓ SCAN (a.k.a. Elevator) and C-SCAN
  Moves back and forth across all tracks
  C-SCAN: handle requests from inner-to-outer, then go back inner tracks directly and handling requests again from inner -to-outer (or reverse)

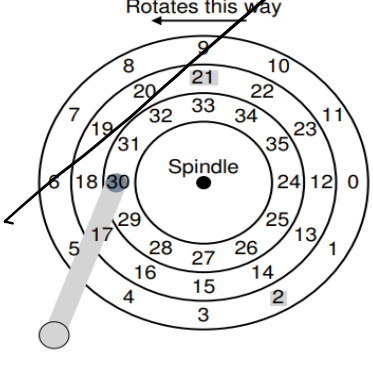
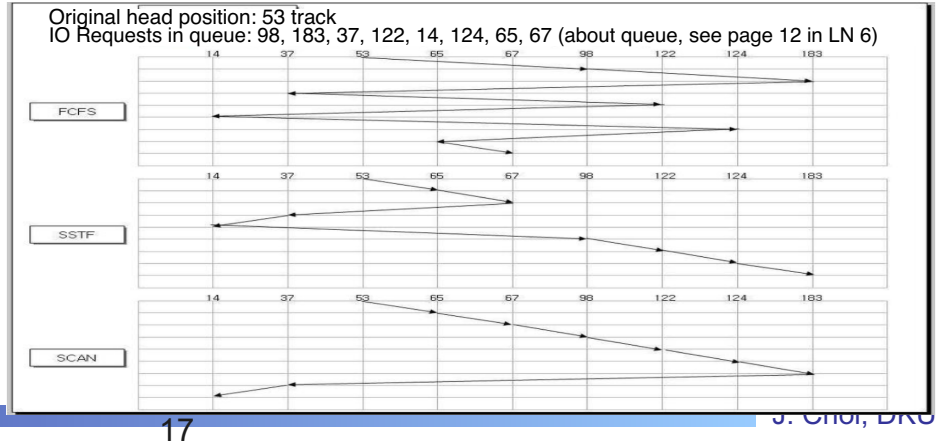


Figure 37.7: **SSTF: Scheduling Requests 21 And 2**



37.5 Disk Scheduling

Examples (cont')

- ✓ SPTF (Shortest Positioning Time First)
  - Consider seek and rotation latency
  - Why? Issues that consider seek only ? not optimal (Figure 37.8)
    - Head position: 30 (sector), Next requests: 16 and 8
    - SSTF: 16 and then 8 ? 1 seek + 5/6 rotation + 1 seek + 2/6 rotation
    - How about 8 and then 16 ? 1 seek (relatively further) + 1/6 rotation + 1 seek + 4/6 rotation
    - Performance depends on disk characteristics (seek vs. rotation)
  - SPTF select a request who has the smallest position time (seek + rotation time)

Other scheduling issues

- ✓ Merge: requests 33, 4, 34, …
- ✓ Anticipatory disk scheduling

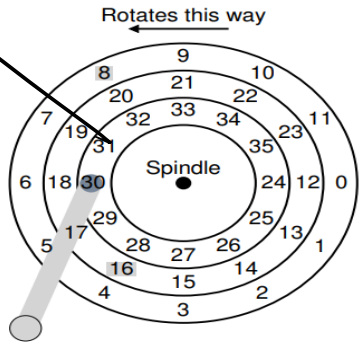


Figure 37.8: **SSTF: Sometimes Not Good Enough**

Chap. 39 Interlude: Files and Directories

39.1 Files and Directories
39.2 File System Interface
39.3 Creating Files
39.4 Reading and Writing Files
39.5 Reading and Writing, But Not Sequentially
39.6 Shared file table entries: fork() and dup()
39.7 Writing immediately with fsync()
39.8 Renaming files
39.9 Getting information about files
39.10 Removing files
39.11 Making Directories
39.12 Reading Directories
39.13 Deleting Directories
39.14 Hard Links
39.15 symbolic Links
39.16 Permission Bits and Access Control Lists
39.17 Making and Mounting a file system

Chap. 39 Interlude: Files and Directories

Computer system

- ✓ Four key abstractions: process (thread), virtual memory, lock, and file
- ✓ Files are in Storage (Hard disk, Solid State Drive)

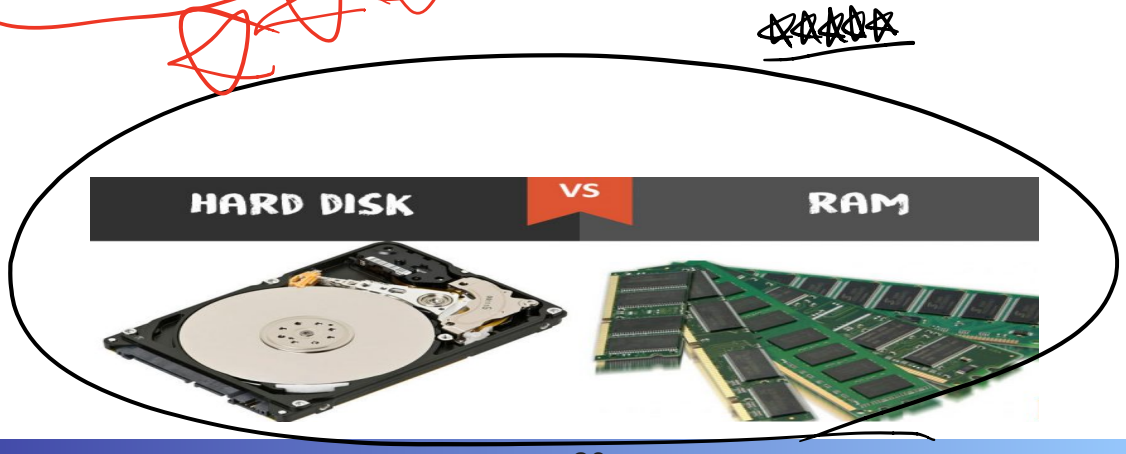
  Storage vs. Memory

  Non-volatility
  - Advantages: Support persistence (store information permanently)
  - Issues: 1) Integrity, 2) Space-efficiency, 3) Consistency, 4) Crash consideration (fault-tolerance), 5) Access control, 6) Security, …

  These issues are managed by a file system
- ✓ How to analysis file system?

  Interface: open, read, write, close, mkdir, link, mount, … (Chapter 39)

  Layout: file, directory, inode, FAT, superblock, … (Chapter 40)



39.1 Files and Directories

File

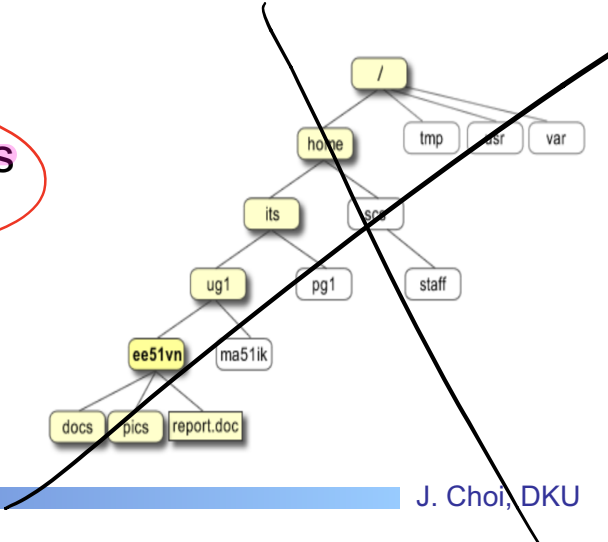
- Definition: A linear array of characters (bytes), stored persistently
  - Each file has various data structure (text, c code, record, multimedia, …)
  - But, OS don't care its content, just treating it as a stream of bytes
- Each file has its name (absolute path, relative path)
- It also has some kind of low-level name in OS (e.g. inode)
  - Like each process has a unique PCB (like program and PCB)

Directory

- A special file that constructs a hierarchy (file hierarchy)
  - Root directory
  - Home directory
  - Working directory
- ~~Contain <file name, inode>~~
  - ~~or low-level name or first disk block~~

Others are also treated as a file

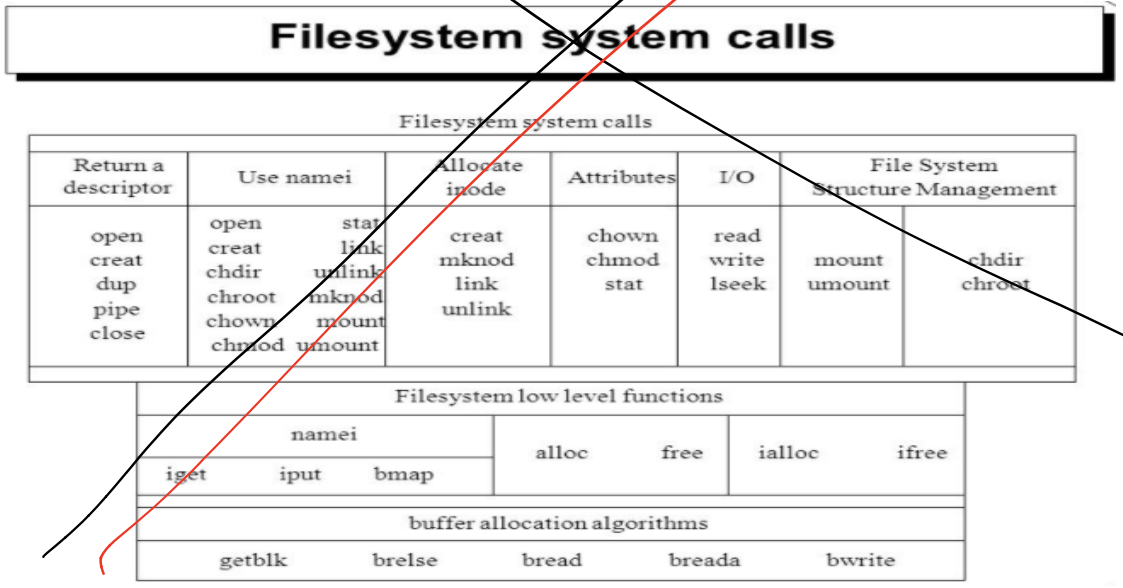
- Device, pipe, socket, and even process



39.2 File System Interfaces

APIs

- ✓ System call: 1) open (return a file descriptor), 2) I/O, 3) attribute, 4) create, 5) name resolution (directory hierarchy traverse), 6) file system management, 7) directory management, …
- ✓ Internals: 1) allocate/free block, 2) allocate/free inode, 3) namei (name-to-inode), 4) buffer related



(Source: http://slideplayer.com/slide/9118590/)

39.3 Creating Files / 39.4 Reading and Writing Files

Create API

- ✓ open() with create flag

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

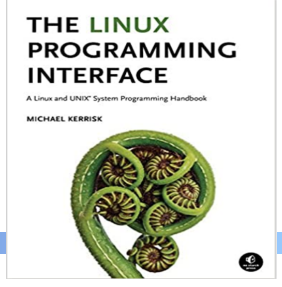
- ✓ creat(): less used (but famous by Ken Thompson's answer about redesigning UNIX)

```
int fd = creat("foo"); // option: add second flag to set permissions
```

Read/Write API

- ✓ read_size = read(fd, buf, request_size);
- ✓ written_size = write(fd, buf, request_size);

  Arguments: 1) fd, 2) buffer that points memory space for data, 3) request size
  Return: read or written size



39.4 Reading and Writing Files

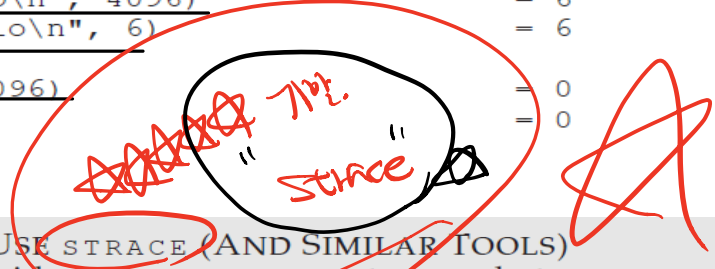
Read and write example

- ✓ Command line viewpoint

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

- ✓ System call viewpoint (u

```
prompt> strace cat foo
...
open("foo", O_RDONLY|O_LARGEFILE)          = 3
read(3, "hello\n", 4096)                   = 6
write(1, "hello\n", 6)                     = 6
hello
read(3, "", 4096)                          = 0
close(3)                                   = 0
...
prompt>
```



TIP: USE STRACE (AND SIMILAR TOOLS)

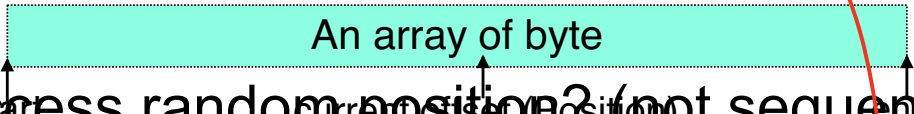
The `strace` tool provides an awesome way to see what programs are up to. By running it, you can trace which system calls a program makes, see the arguments and return codes, and generally get a very good idea of what is going on.

The tool also takes some arguments which can be quite useful. For example, `-f` follows any fork'd children too; `-t` reports the time of day at each call; `-e trace=open,close,read,write` only traces calls to those system calls and ignores all others. There are many more powerful flags — read the man pages and find out how to harness this wonderful tool.

39.5 Reading and Writing, But Not Sequentially

Conventional accessing mechanism for a file

- ✓ Sequential
- ✓ From the begin, increasing the offset while reading or writing



How to access random position? (not sequentially)

- ✓ lseek()

Arguments: 1) fd, 2) relative offset from whence, 3) reference point

- Whence: SEEK_SET, SEEK_CUR, SEEK_END

Explicit update the current offset (c.f. read/write: implicit update)
Do not confuse lseek() with disk seek :-)

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

☞ Also do not confuse process and processor

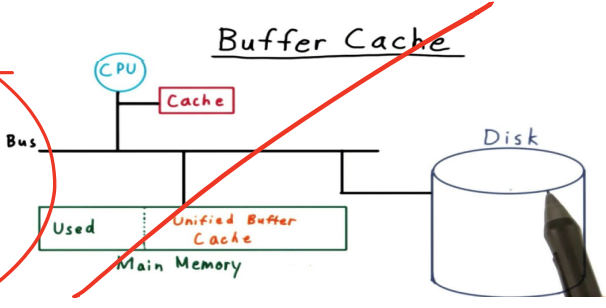
39.6 Writing Immediately with fsync()

Performance consideration for write

- ✓ Write to DRAM vs Disk 100ns vs 10,000,000ns (10ms)
- ✓ Delayed write
  - Write data into DRAM (called buffer or page cache) and set them dirty
  - Later write all dirty data into disk in a clustering fashion (5 or 30 seconds periodically)
  - Write grouping and write reordering indeed enhance performance
  - Synchronous vs Asynchronous

Concern of delayed write

- ✓ Durability
  - User think his/her data is permanent but not in actuality
- ✓ How to guarantee durability
  - fsync() system call




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

39.7 Renaming Files / 39.9 Removing Files

Change a file name

- ✓ Command line viewpoint

rename(__, __)

- ✓ System call viewpoint (editor example)

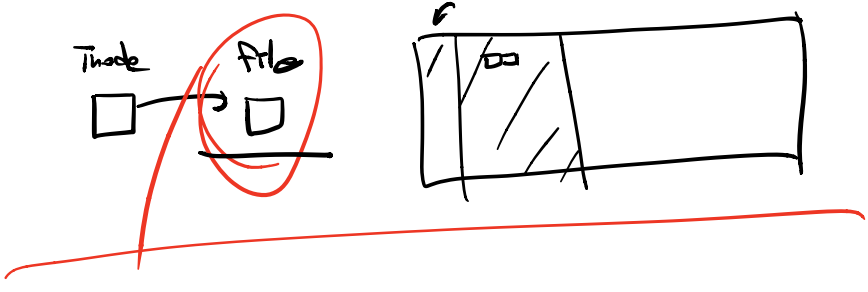
```
prompt> mv foo bar
```

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

Remo

- ✓ A

unlink(file name)




```
prompt> strace rm foo
...
unlink("foo")                           = 0
...
```

☞ Why not remove() or delete() instead of unlink()? Then, what is link()?

39.8 Getting Information about Files

Contents in a file system

- ✓ Two types of data in file system: User data vs. Metadata
  - User data (or just data): data written by users
  - Metadata: data written by a file system for managing files (in inode) and file system (in superblock)
- ✓ API to see the metadata for a certain file
  - stat(file_name, struct stat)
  - fstat(fd, struct stat)

```
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 */
};
```

```
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/ remzi) 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
```

39.10 Making Directories / 39.12 Deleting Directories

API for making directory

- ✓ mkdir(name, permission)

```
prompt> strace mkdir foo
...
mkdir("foo", 0777)                      = 0
...
prompt>
```

- ✓ After

API for de

- ✓ rmdir(
- ✓ We n

```
prompt> ls -a
./   ../
prompt> ls -al
total 8
drwxr-x---  2 remzi remzi    6 Apr 30 16:17 ./
drwxr-x--- 26 remzi remzi 4096 Apr 30 16:17 ../
```

39.11 Reading Directories

APIs for reading directory

- opendir(dp), readdir(dp), closedir(dp)
  - "ls": like the below example (c.f. "ls –l": readdir() + stat())

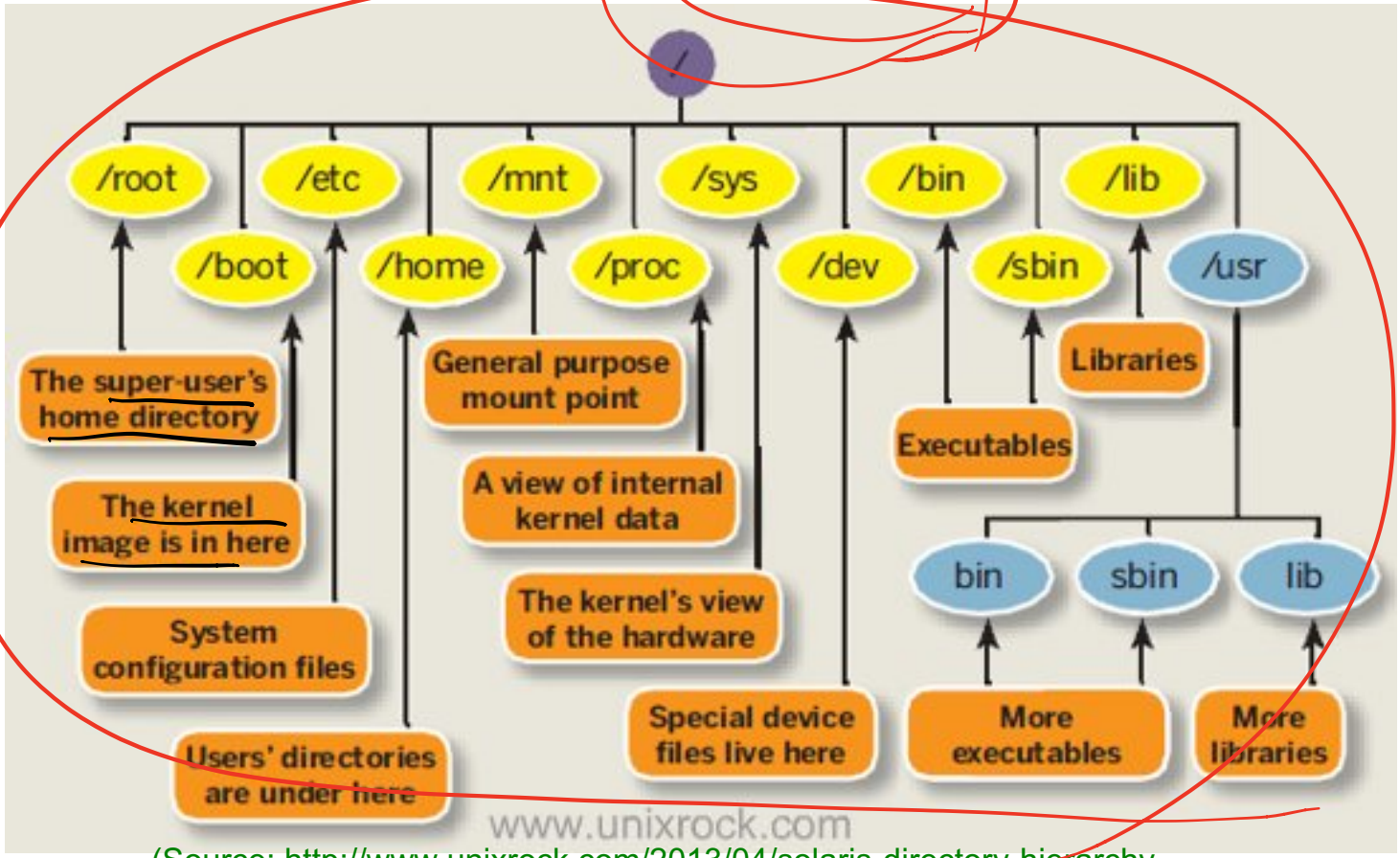
```
int main(int argc, char *argv[]) {
    DIR *dp = opendir(".");
    assert(dp != NULL);
    struct dirent *d;
    while ((d = readdir(dp)) != NULL) {
        printf("%d %s\n", (int) d->d_ino, d->d_name);
    }
    closedir(dp);
    return 0;
}
```

```
struct dirent {
    char           d_name[256]; /* filename */
    ino_t          d_ino;       /* inode number */
    off_t          d_off;       /* offset to the next dirent */
    unsigned short d_reclen;    /* length of this record */
    unsigned char  d_type;      /* type of file */
};
```

☞ Why there is no writedir()?

39.11 Reading Directories

Directory name convention



(Source: http://www.unixrock.com/2013/04/solaris-directory-hierarchy.html)

39.13 Hard Links

Link

- ✓ Make another file name to access an existing file
  - Connect a file name with an inode
- ✓ Command line viewpoint
  - Either file or file2
- ✓ API
- ✓ After

```
prompt> echo hello > file
prompt> cat file
hello
prompt> ln file file2
prompt> cat file2
hello
```

```
prompt> ls -i file file2
67158084 file
67158084 file2
prompt>
```

link api ←

- ✓ Link count
  - Delete data when link count is 0

unlink ←

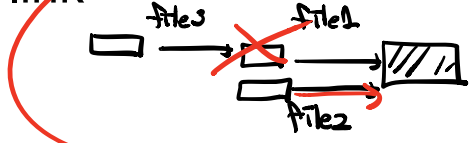
```
prompt> rm file
removed 'file'
prompt> cat file2
hello
```

```
prompt> echo hello > file
prompt> stat file
... Inode: 67158084    Links: 1 ...
prompt> ln file file2
prompt> stat file
... Inode: 67158084    Links: 2 ...
prompt> stat file2
... Inode: 67158084    Links: 2 ...
prompt> ln file2 file3
prompt> stat file
... Inode: 67158084    Links: 3 ...
prompt> rm file
prompt> stat file2
... Inode: 67158084    Links: 2 ...
prompt> rm file2
prompt> stat file3
... Inode: 67158084    Links: 1 ...
prompt> rm file3
```

39.14 Symbolic Links

Link

- ✓ Hard link: share inode number
  - Create a new file name and share the existing inode
- ✓ Symbolic link (Soft link): different inode number, but its data is the linked file name
  - Create not only a new file name but also a new inode (set it as a symbolic link)
  - Can link between different file systems, Can link to a directory
- ✓ Dangling reference in symbolic link



기본 단점

```
oslab@osLab: ~/os_test
oslab@osLab:~/os_test$ ls -al
합계 8
drwxrwxr-x  2 oslab oslab 4096  4월 23 12:09 .
drwxr-xr-x 22 oslab oslab 4096  4월 23 12:03 ..
oslab@osLab:~/os_test$
oslab@osLab:~/os_test$ echo "hello world" > file1
oslab@osLab:~/os_test$ ln file1 file2
oslab@osLab:~/os_test$ ln -s file1 file3
oslab@osLab:~/os_test$
oslab@osLab:~/os_test$ ls -ail
합계 16
8297 drwxrwxr-x  2 oslab oslab 4096  4월 23 12:09 .
8196 drwxr-xr-x 22 oslab oslab 4096  4월 23 12:03 ..
 380 -rw-rw-r--  2 oslab oslab   12  4월 23 12:09 file1
 380 -rw-rw-r--  2 oslab oslab   12  4월 23 12:09 file2
 386 lrwxrwxrwx  1 oslab oslab    5  4월 23 12:09 file3 -> file1
oslab@osLab:~/os_test$
oslab@osLab:~/os_test$ rm file1
oslab@osLab:~/os_test$
oslab@osLab:~/os_test$ cat file2
hello world
oslab@osLab:~/os_test$ cat file3
cat: file3: 그런 파일이나 디렉터리가 없습니다
oslab@osLab:~/os_test$
oslab@osLab:~/os_test$ ls -i
380 file2  386 file3
oslab@osLab:~/os_test$
```

39.15 Making and Mounting a File System

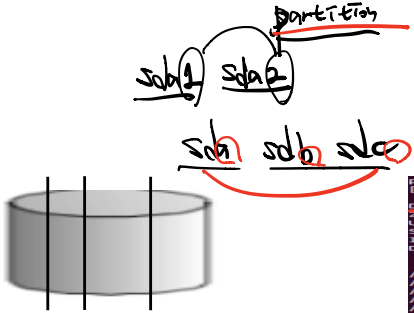
File system

- ✓ Make a file system
  - Assemble directories and files
  - Related metadata: superblock, bitmap, … (main topic in chapter 40)
  - Command: mkfs
    - Make an empty file system (only root directory) in a disk partition

How to

- ✓ Example
  - Partition
  - Ext2/3/4

```
howtogeek@ubuntu: ~
howtogeek@ubuntu:~$ sudo mkfs
mkfs          mkfs.ext2     mkfs.ext4dev  mkfs.ntfs
mkfs.bfs      mkfs.ext3     mkfs.minix    mkfs.vfat
mkfs.cramfs   mkfs.ext4     mkfs.msdos
howtogeek@ubuntu:~$ sudo mkfs.ext4 /dev/sda5
```



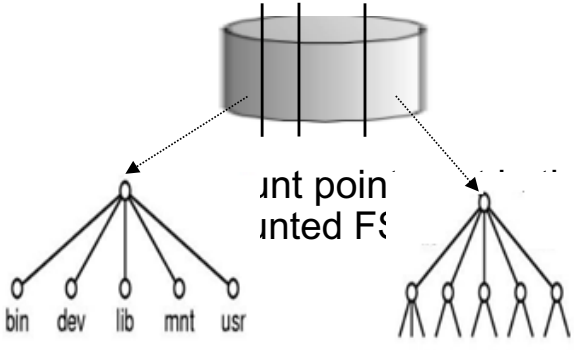
```
prltam@PPH:~$ sudo fdisk -l
[sudo] password for prltam:

Disk /dev/sda: 500.1 GB, 500107862016 bytes
255 heads, 63 sectors/track, 60801 cylinders, total 976773168 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x17078a5

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *        2048   976568831   488283392    7  HPFS/NTFS/exFAT
/dev/sda2       976568832  2931777343   977604256    7  HPFS/NTFS/exFAT
/dev/sda3      2931777344  8791116287  2929969472    7  HPFS/NTFS/exFAT
/dev/sda4      8791118334   976773119   488827393    5  Extended
/dev/sda5      8791118336   888881151     4881408   82  Linux swap / Solaris
/dev/sda6       888883200   976773119    43944960   83  Linux
prltam@PPH:~$
```

Why multiple partitions?

39.15 Making and Mounting a File System

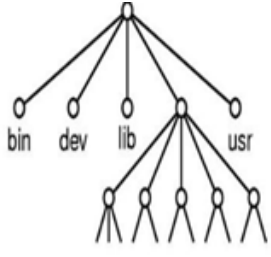
File system

- ✓ Mount
  - Make a file system visible to users
  - Connect multiple file systems within the uniform directory tree
    - mount arguments: 1) FS type, 2) partition, 3) mount point



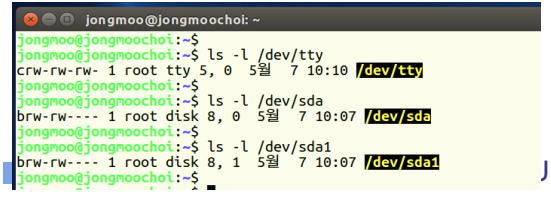
Before mount

$mount –t ext3 /dev/sda4 /mnt



After mount

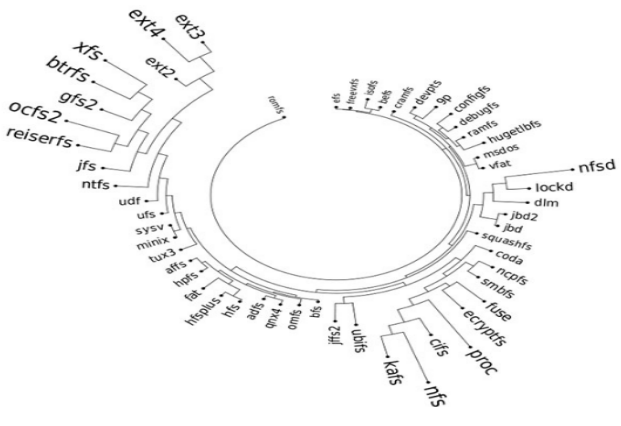
- Simple question to take attendance: The right snapshot is what I check the attributes of /dev/tty /dev/sda and /dev/sda1 in ubuntu. Explain meaning of attributes such as “b”, “c”, ”rw” “5”, “8”, “1” and so on (until 6 PM, May 14th)

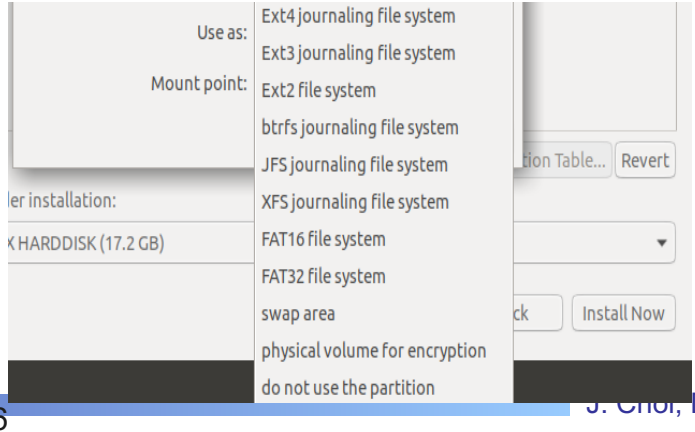


Chap. 40 File System Implementation

Objective of this chapter

- ✓ A variety of file systems
  UFS, FFS, EXT2/3/4, JFS, LFS, NTFS, F2FS, FUSE, RAMFS, NFS, AFS, ZFS, GFS, ….
- ✓ Make a new file system: called VSFS(Very Simple File System)
  Simplified version of UFS (Unix File System)
  1) On-disk structures: inode, bitmap, directory, …
  2) Access method: read, write, …
  3) Various policies: cache, delayed write, …
- ✓ More complex file systems ? next chapters



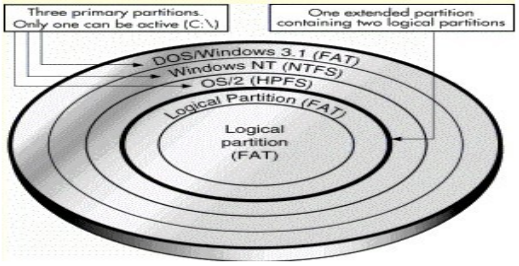


40.1 The Way to Think / 40.2 Overall Organization

Disk

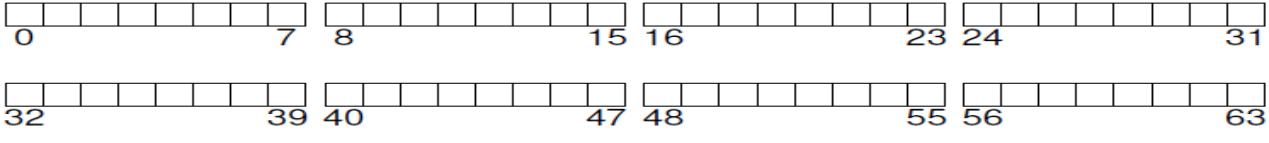
- ✓ Consist of partitions
- ✓ A file system is created in each partition

Partition




```
Disk /dev/sda: 320.1 GB, 320072933376 bytes
255 heads, 63 sectors/track, 38913 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0xaa692010

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *           1        7681    61690880    7  HPFS/NTFS
/dev/sda2            7681       14182    52219904    7  HPFS/NTFS
/dev/sda3           14182       20556    51200000    7  HPFS/NTFS
/dev/sda4           20556       38913   147453953    f  W95 Ext'd (LBA)
/dev/sda5           20556       32030    92160000    7  HPFS/NTFS
/dev/sda6           34324       34770     3583999+  82  Linux swap / Solaris
/dev/sda7           34771       38913    33276928    7  HPFS/NTFS
/dev/sda8           32030       34323    18423808   83  Linux

Partition table entries are not in disk order
```

- ✓ Cons
- ✓ User loc page)
- ✓ Assume a partition having 64 disk blocks (or simply blocks)

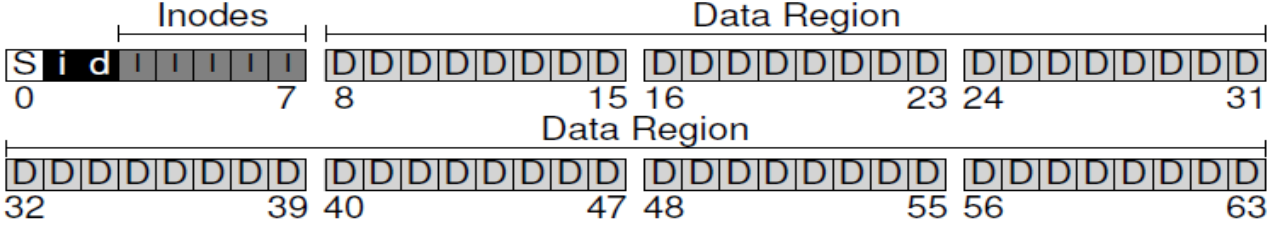


☞ Now consider what data structures are required for making a FS ?

40.2 Overall Organization

Layout of a file system

- ✓ User data: 8 ~ 63 blocks (can be dynamically adjusted)
  - Data written by users
- ✓ Inode: 3~7 blocks
  - Metadata for managing files (one per a file)
  - Inode size = 256B ? 16 inodes per a block ? 5 blocks for inode ? total 80 files can be created
- ✓ Bitmap: 1~2 blocks
  - Metadata for managing free space (allocation structure)
  - Two bitmaps: one for data blocks and the other for inodes
- ✓ Superblock: 0 blocks
  - Metadata for managing a file system (one per a file system)
    - Information: how many data blocks, inodes, where they begin, …
  - Used during a mount function



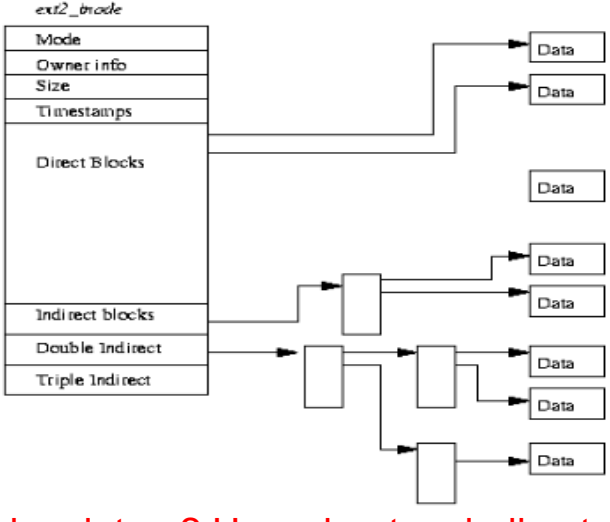
40.3 File Organization: The inode

How to manage metadata for a file

- ✓ inode (index node)
  - File information such as mode, uid, size, time, link count, blocks, …
    - Can be accessed using stat()
  - Locations of User data blocks ? Multi-Level index (or imbalanced tree)
    - Direct block pointers (10 or 12 or 15), Single/Double/Triple indirect block pointers(1/1/1)
    - Benefit: Fast for a short file and Big size support for a large file
- ✓ Other approach: FAT (linked based), Extent-based, Log-based, ..

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?
2	gid	which group does this file belong to?
2	links_count	how many hard links are there to this file?
4	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)
4	generation	file version (used by NFS)
4	file_acl	a new permissions model beyond mode bits
4	dir_acl	called access control lists

Figure 40.1: **Simplified Ext2 Inode**

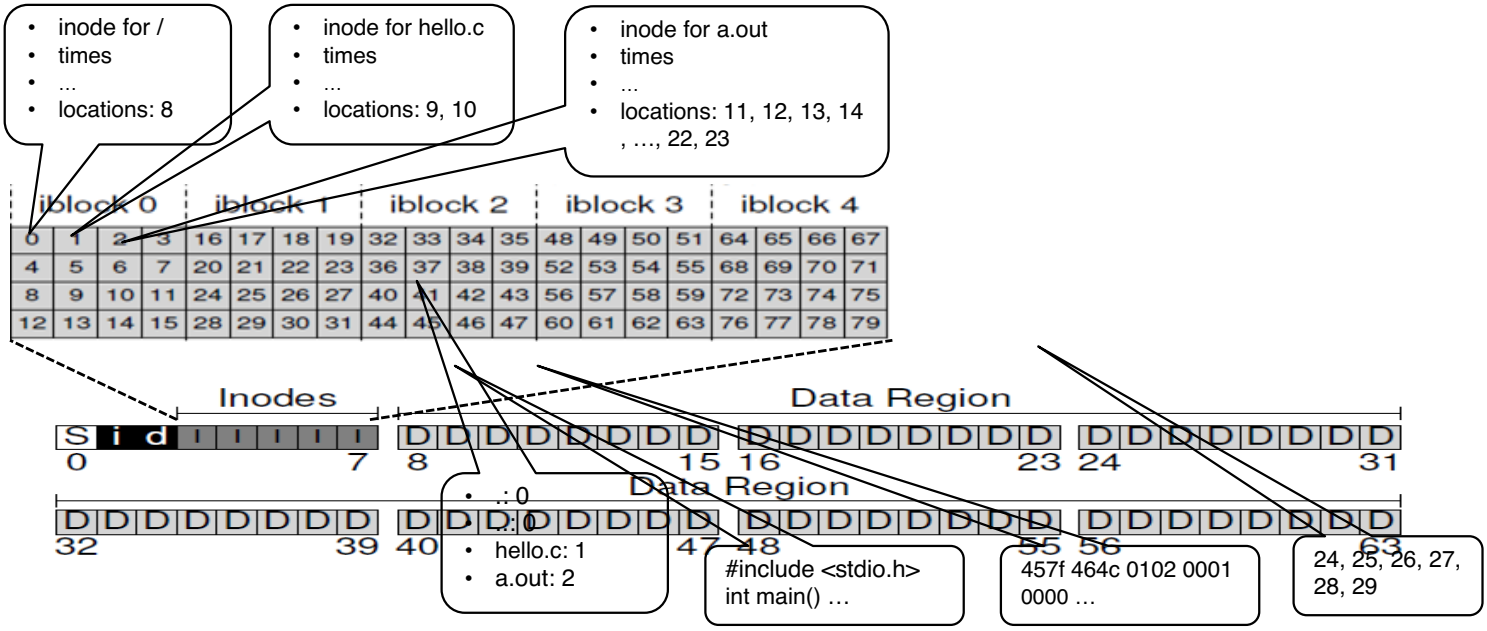


☞ How large size can be supported by direct block pointers? How about an indirect pointer?

40.3 File Organization: The inode

inode manipulation example (assume 12 direct blocks)

- ✓ When we create a new file (named hello.c whose size is 7KB) in a root directory?
- ✓ Then, we compile it? (a.out whose size is 70KB)

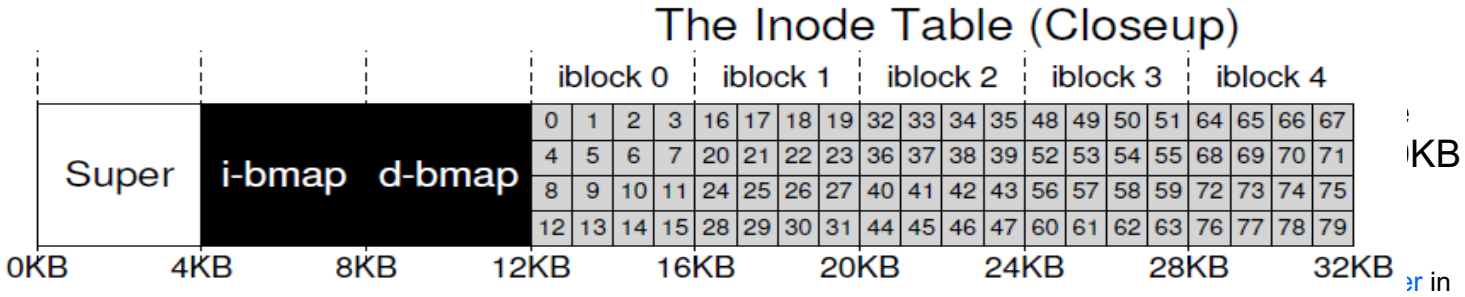


40.3 File Organization: The inode

Find a location: inode and data

- ✓ How to find the location of a inode?

  Directory entry: <file name, i_number>
  i_number: index in inode region (inode table)



the inode (multi-level index), 4) remainder is used as the offset in the disk block

40.4 Directory Organization / 40.5 Free Space Mgmt

Directory

- ✓ Basically, a list of pairs <file name, inode number>
- ✓ For fast search, add the file name length and record length (total bytes including left over space)

inum	reclen	strlen	name
5	4	2	.
2	4	3	..
12	4	4	foo
13	4	4	bar
24	8	7	foobar

- ✓ Can [illegible] directory (e.g. B-tree in XFS)

Free space

- ✓ Bitmap: one bit per disk block (or inode), indicating whether it is free or used
- ✓ Alternative approach: free-list, tree, …
- ✓ Pre-allocation: allocate free disk blocks in a batch manner ? less overhead, contiguous allocation, …

40.6 Access Paths: Reading and Writing

Reading a file from disk

- ✓ open a file “/foo/bar” whose size is 12KB, read data and close it
- ✓ Timeline

	data bitmap	inode bitmap	root inode	foo inode	bar inode	root data	foo data	bar data[0]	bar data[1]	bar data[2]
open(bar)			read			read				
				read						
							read			
					read					
read()					read					
								read		
					write					
read()					read					
									read	
					write					
read()					read					
										read
					write					

Figure 40.3: **File Read Timeline (Time Increasing Downward)**

40.6 Access Paths: Reading and Writing

Writing a file into disk

- ✓ Create a file "/foo/bar", write data and close it
- ✓ Timeline

	data bitmap	inode bitmap	root inode	foo inode	bar inode	root data	foo data	bar data[0]	bar data[1]	bar data[2]
			read			read				
				read			read			
create (/foo/bar)		read write								
							write			
					read write					
				write						
					read					
write()	read write							write		
					write					
					read					
write()	read write								write	
					write					
					read					
write()	read write									write
					write					

Figure 40.4: **File Creation Timeline (Time Increasing Downward)**

40.7 Caching and Buffering

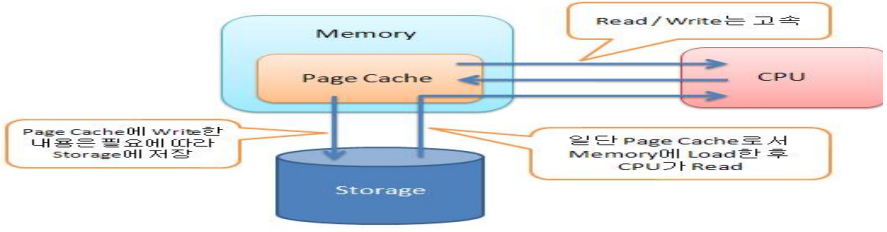
Issues

- ✓ Disk is too slow.

> THE CRUX: HOW TO REDUCE FILE SYSTEM I/O COSTS
> Even the simplest of operations like opening, reading, or writing a file incurs a huge number of I/O operations, scattered over the disk. What can a file system do to reduce the high costs of doing so many I/Os?

Solutions

- ✓ Caching
  - Caching directories (e.g. / inode, / data, current directory, …) in DRAM
  - Caching recently used file's inodes and data in DRAM
  - Management: LRU replacement policy, dynamic cache size management
- ✓ Write buffering (Delayed write)
  - Consolidate several writes into a single one: e.g.) d-bitmap
  - Schedule multiple writes so that they have less seek overhead: e.g.) bar data
  - Avoid writes: e.g.) temporary file (create and delete immediately)
  - Concern: Data loss due to power fault or crash ? fsync() or direct I/O

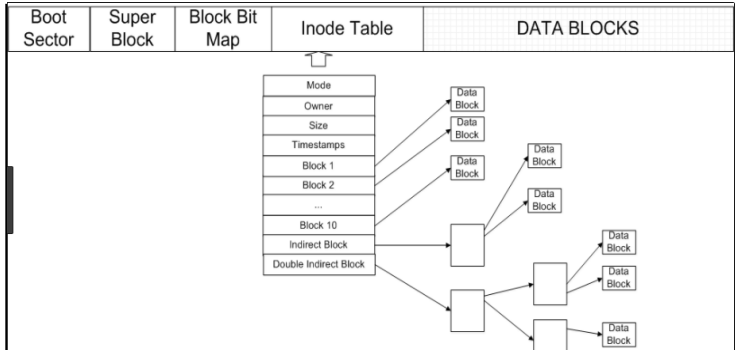


(Source: http://www.atmarkit.co.jp/ait/articles/0810/01/news134_2.html")

40.8 Summary

File system

- ✓ Interface
  - open(), read(), write(), …
  - mkdir(), readdir(), …
  - mount(), mknod(), …
- ✓ Layout
  - Data blocks
  - Inode, Bitmap, Superblock
  - Boot block



Importance of mental model

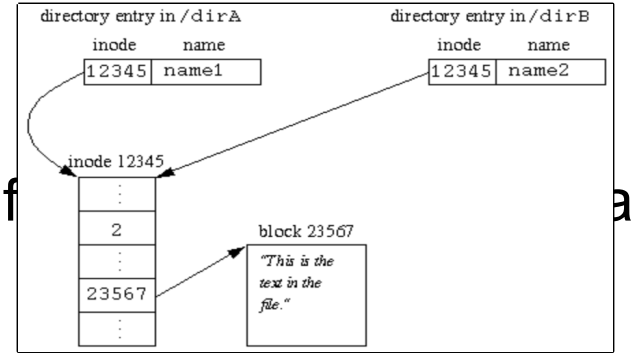
- Simple question to take attendance: In 40 page, we discuss how a file is connected into disk blocks using inode. In this example, 1) which disk block is accessed when we want to read the a.out file with the current_offset = 10000? 2) which disk block is accessed with the current_offset = 60000 (hint: use quotient discussed in 41 page). (until 6 PM, May 20th)

ASIDE: MENTAL MODELS OF FILE SYSTEMS

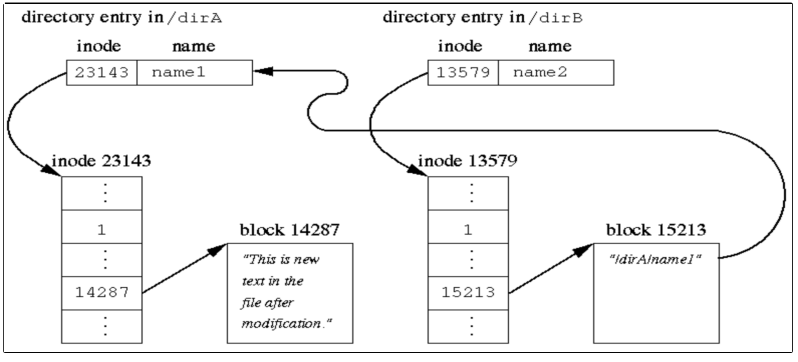
As we've discussed before, mental models are what you are really trying to develop when learning about systems. For file systems, your mental model should eventually include answers to questions like: what on-disk structures store the file system's data and metadata? What happens when a process opens a file? Which on-disk structures are accessed during a read or write? By working on and improving your mental model, you develop an abstract understanding of what is going on, instead of just trying to understand the specifics of some file-system code (though that is also useful, of course!).

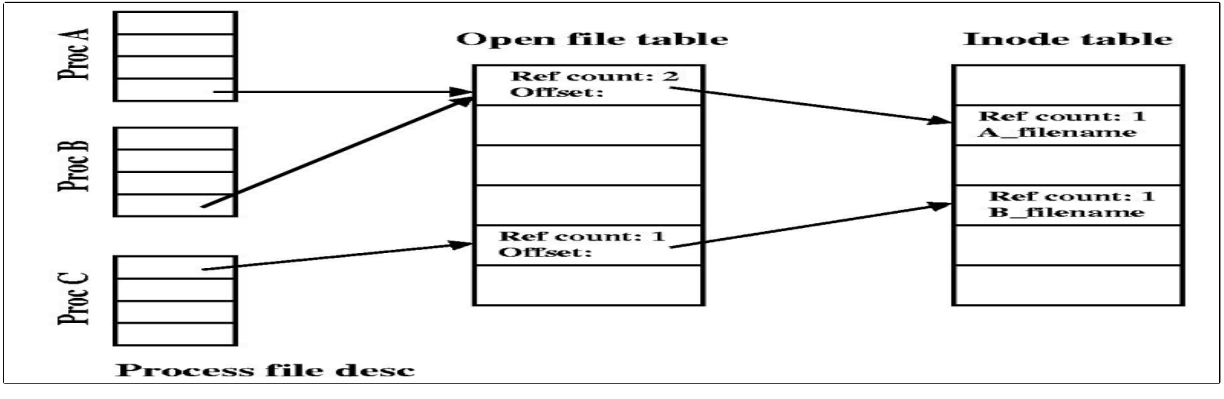
Appendix

Hard link vs. Symbolic link(Soft link)



f able

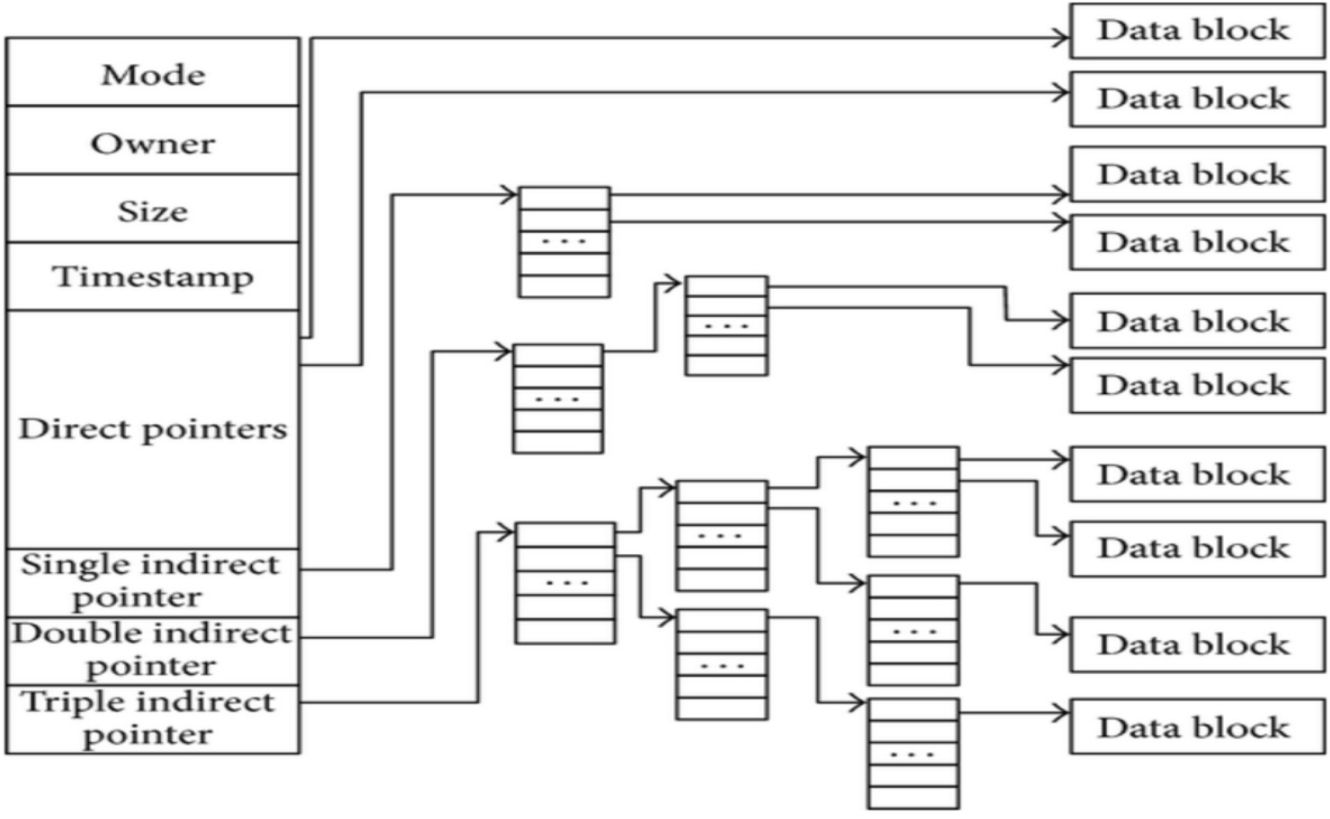




(Source: http://classque.cs.utsa.edu/classes/cs3733/notes/USP-05.html)

Appendix

inode internals



(Source: https://www.researchgate.net/figure/The-architecture-of-an-inode-in-EXT3-file-system_fig2_258396310)