

CS162  
Operating Systems and  
Systems Programming  
Lecture 18

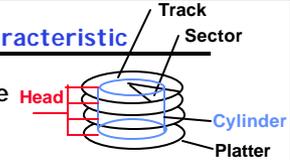
File Systems, Naming, and Directories

April 3, 2006

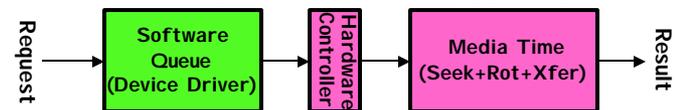
Prof. Anthony D. Joseph

<http://inst.eecs.berkeley.edu/~cs162>

Review: Magnetic Disk Characteristic



- Cylinder: all the tracks under the head at a given point on all surface
- Read/write data is a three-stage process:
  - Seek time: position the head/arm over the proper track (into proper cylinder)
  - Rotational latency: wait for the desired sector to rotate under the read/write head
  - Transfer time: transfer a block of bits (sector) under the read-write head
- **Disk Latency = Queuing Time + Controller time + Seek Time + Rotation Time + Xfer Time**



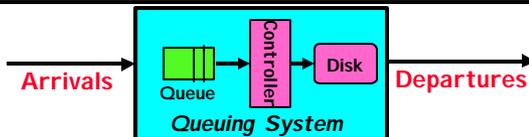
- **Highest Bandwidth:**
  - transfer large group of blocks sequentially from one track

4/3/06

Joseph CS162 ©UCB Spring 2006

Lec 18.2

Review: Introduction to Queuing Theory



- What about queuing time??
  - Let's apply some queuing theory
  - Queuing Theory applies to long term, steady state behavior  $\lambda$  Arrival rate = Departure rate
- Little's Law:
  - Mean # tasks in system = arrival rate x mean response time**
  - Observed by many, Little was first to prove
  - Simple interpretation: you should see the same number of tasks in queue when entering as when leaving.
- Applies to any system in equilibrium, as long as nothing in black box is creating or destroying tasks
  - **Typical queuing theory doesn't deal with transient behavior, only steady-state behavior**

4/3/06

Joseph CS162 ©UCB Spring 2006

Lec 18.3

Goals for Today

- Finishing Disk Performance
  - Hardware performance parameters
  - Queuing Theory
- File Systems
  - Structure, Naming, Directories

Note: Some slides and/or pictures in the following are adapted from slides ©2005 Silberschatz, Galvin, and Gagne. Many slides generated from my lecture notes by Kubiatowicz.

4/3/06

Joseph CS162 ©UCB Spring 2006

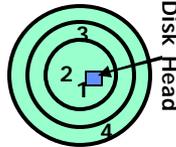
Lec 18.4

## Disk Scheduling

- Disk can do only one request at a time; What order do you choose to do queued requests?



- FIFO Order
  - Fair among requesters, but order of arrival may be to random spots on the disk ⇒ Very long seeks
- SSTF: Shortest seek time first
  - Pick the request that's closest on the disk
  - Although called SSTF, today must include rotational delay in calculation, since rotation can be as long as seek
  - Con: SSTF good at reducing seeks, but may lead to starvation
- SCAN: Implements an Elevator Algorithm: take the closest request in the direction of travel
  - No starvation, but retains flavor of SSTF
- C-SCAN: Circular-Scan: only goes in one direction
  - Skips any requests on the way back
  - Fairer than SCAN, not biased towards pages in middle



4/3/06

Joseph CS162 ©UCB Spring 2006

Lec 18.5

## Administrivia

4/3/06

Joseph CS162 ©UCB Spring 2006

Lec 18.6

## Building a File System

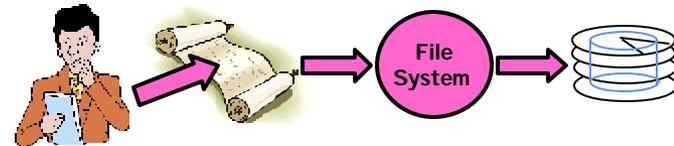
- **File System:** Layer of OS that transforms block interface of disks (or other block devices) into Files, Directories, etc.
- File System Components
  - Disk Management: collecting disk blocks into files
  - Naming: Interface to find files by name, not by blocks
  - Protection: Layers to keep data secure
  - Reliability/Durability: Keeping of files durable despite crashes, media failures, attacks, etc
- User vs. System View of a File
  - User's view:
    - » Durable Data Structures
  - System's view (system call interface):
    - » Collection of Bytes (UNIX)
    - » Doesn't matter to system what kind of data structures you want to store on disk!
  - System's view (inside OS):
    - » Collection of blocks (a block is a logical transfer unit, while a sector is the physical transfer unit)
    - » Block size <sup>3</sup> sector size; in UNIX, block size is 4KB

4/3/06

Joseph CS162 ©UCB Spring 2006

Lec 18.7

## Translating from User to System View



- What happens if user says: give me bytes 2—12?
  - Fetch block corresponding to those bytes
  - Return just the correct portion of the block
- What about: write bytes 2—12?
  - Fetch block
  - Modify portion
  - Write out Block
- Everything inside File System is in whole size blocks
  - For example, `getc()`, `putc()` ⇒ buffers something like 4096 bytes, even if interface is one byte at a time
- From now on, file is a collection of blocks

4/3/06

Joseph CS162 ©UCB Spring 2006

Lec 18.8

## Disk Management Policies

- Basic entities on a disk:
  - **File**: user-visible group of blocks arranged sequentially in logical space
  - **Directory**: user-visible index mapping names to files (next lecture)
- Access disk as linear array of blocks. Two Options:
  - Identify blocks as vectors [cylinder, surface, sector]. Sort in cylinder-major order. Not used much anymore.
  - **Logical Block Addressing (LBA)**. Every block has integer address from zero up to max number of cylinders.
  - Controller translates from address  $P$  physical position
    - » First case: OS/BIOS must deal with bad blocks
    - » Second case: hardware shields OS from structure of disk
- Need way to track free disk blocks
  - Link free blocks together  $P$  too slow today
  - Use bitmap to represent free space on disk
- Need way to structure files: **File Header**
  - Track which blocks belong at which offsets within the logical file structure
  - **Optimize placement of files disk blocks to match access and usage patterns**

4/3/06

Joseph CS162 ©UCB Spring 2006

Lec 18.9

## Designing the File System: Access Patterns

- How do users access files?
  - Need to know type of access patterns user is likely to throw at system
- Sequential Access: bytes read in order ("give me the next X bytes, then give me next, etc")
  - Almost all file access are of this flavor
- Random Access: read/write element out of middle of array ("give me bytes  $i-j$ ")
  - Less frequent, but still important. For example, virtual memory backing file: page of memory stored in file
  - Want this to be fast - don't want to have to read all bytes to get to the middle of the file
- Content-based Access: ("find me 100 bytes starting with JOSEPH")
  - Example: employee records - once you find the bytes, increase my salary by a factor of 2
  - Many systems don't provide this; instead, databases are built on top of disk access to index content (requires efficient random access)

4/3/06

Joseph CS162 ©UCB Spring 2006

Lec 18.10

## Designing the File System: Usage Patterns

- Most files are small (for example, .login, .c files)
  - A few files are big - nachos, core files, etc.; the nachos executable is as big as all of your .class files combined
  - However, most files are small - .class's, .o's, .c's, etc.
- Large files use up most of the disk space and bandwidth to/from disk
  - May seem contradictory, but a few enormous files are equivalent to an immense # of small files
- Although we will use these observations, beware usage patterns:
  - Good idea to look at usage patterns: beat competitors by optimizing for frequent patterns
  - Except: changes in performance or cost can alter usage patterns. Maybe UNIX has lots of small files because big files are really inefficient?
- Digression, danger of predicting future:
  - In 1950's, marketing study by IBM said total worldwide need for computers was 7!
  - Company (that you haven't heard of) called "GenRad" invented oscilloscope; thought there was no market, so sold patent to Tektronix (bet you have heard of them!)

4/3/06

Joseph CS162 ©UCB Spring 2006

Lec 18.11

## How to organize files on disk

- Goals:
  - Maximize sequential performance
  - Easy random access to file
  - Easy management of file (growth, truncation, etc)
- First Technique: Continuous Allocation
  - Use continuous range of blocks in logical block space
    - » Analogous to base+bounds in virtual memory
    - » User says in advance how big file will be (disadvantage)
  - Search bit-map for space using best fit/first fit
    - » What if not enough contiguous space for new file?
  - File Header Contains:
    - » First sector/LBA in file
    - » File size (# of sectors)
  - Pros: Fast Sequential Access, Easy Random access
  - Cons: **External Fragmentation/Hard to grow files**
    - » Free holes get smaller and smaller
    - » Could compact space, but that would be *really* expensive
- Continuous Allocation used by IBM 360
  - Result of allocation and management cost: People would create a big file, put their file in the middle

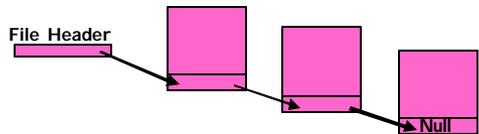
4/3/06

Joseph CS162 ©UCB Spring 2006

Lec 18.12

## Linked List Allocation

- Second Technique: Linked List Approach
  - Each block, pointer to next on disk



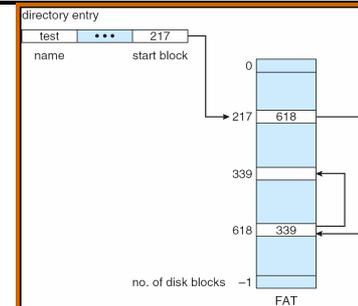
- Pros: Can grow files dynamically, Free list same as file
- Cons: **Bad Sequential Access** (seek between each block), **Unreliable** (lose block, lose rest of file)
- Serious Con: Bad random access!!!!
- **Technique originally from Alto** (First PC, built at Xerox)
  - » No attempt to allocate contiguous blocks

4/3/06

Joseph CS162 ©UCB Spring 2006

Lec 18.13

## Linked Allocation: File-Allocation Table (FAT)



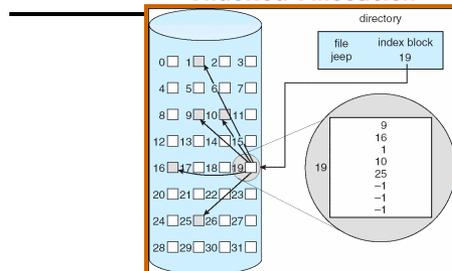
- MSDOS links pages together to create a file
  - Links not in pages, but in the File Allocation Table (FAT)
    - » FAT contains an entry for each block on the disk
    - » FAT Entries corresponding to blocks of file linked together
  - Access properties:
    - » Sequential access expensive unless FAT cached in memory
    - » Random access expensive always, but *really* expensive if FAT not cached in memory

4/3/06

Joseph CS162 ©UCB Spring 2006

Lec 18.14

## Indexed Allocation



- Third Technique: Indexed Files (Nachos, VMS)
  - System Allocates file header block to hold array of pointers big enough to point to all blocks
    - » User pre-declares max file size;
  - Pros: Can easily grow up to space allocated for index
    - Random access is fast
  - Cons: **Clumsy to grow file bigger than table size**  
**Still lots of seeks: blocks may be spread over disk**

4/3/06

Joseph CS162 ©UCB Spring 2006

Lec 18.15

## Multilevel Indexed Files (UNIX 4.1)

- Multilevel Indexed Files: Like multilevel address translation (from UNIX 4.1 BSD)
  - Key idea: efficient for small files, but still allow big files
  - File header contains 13 pointers
    - » Fixed size table, pointers not all equivalent
    - » This header is called an "inode" in UNIX
  - File Header format:
    - » First 10 pointers are to data blocks
    - » Block 11 points to "indirect block" containing 256 blocks
    - » Block 12 points to "doubly indirect block" containing 256 indirect blocks for total of 64K blocks
    - » Block 13 points to a triply indirect block (16M blocks)
- Discussion
  - Basic technique places an upper limit on file size that is approximately 16Gbytes
    - » Designers thought this was bigger than anything anyone would need. Much bigger than a disk at the time...
    - » Fallacy: today, EOS producing 2TB of data per day
  - Pointers get filled in dynamically: need to allocate indirect block only when file grows > 10 blocks.
    - » On small files, no indirection needed

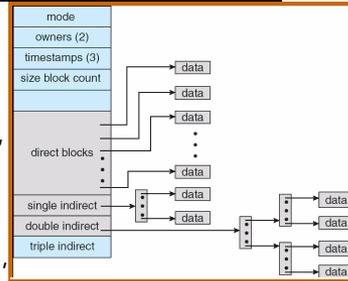
4/3/06

Joseph CS162 ©UCB Spring 2006

Lec 18.16

### Example of Multilevel Indexed Files

- Sample file in multilevel indexed format:
  - How many accesses for block #23? (assume file header accessed on open)
    - » Two: One for indirect block, one for data
  - How about block #5?
    - » One: One for data
  - Block #340?
    - » Three: double indirect block, indirect block, and data



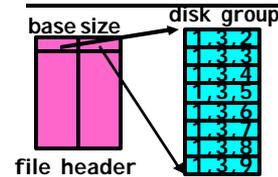
- UNIX 4.1 Pros and cons
  - Pros: Simple (more or less)  
Files can easily expand (up to a point)  
Small files particularly cheap and easy
  - Cons: Lots of seeks  
Very large files must read many indirect block (four I/Os per block!)

4/3/06

Joseph CS162 ©UCB Spring 2006

Lec 18.17

### File Allocation for Cray-1 DEMOS



Basic Segmentation Structure:  
Each segment contiguous on disk

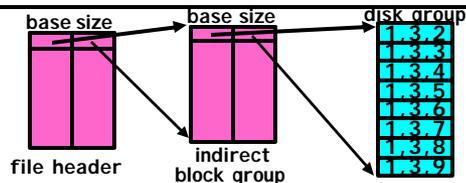
- DEMOS: File system structure similar to segmentation
  - Idea: reduce disk seeks by
    - » using contiguous allocation in normal case
    - » but allow flexibility to have non-contiguous allocation
  - Cray-1 had 12ns cycle time, so CPU:disk speed ratio about the same as today (a few million instructions per seek)
- Header: table of base & size (10 "block group" pointers)
  - Each block chunk is a contiguous group of disk blocks
  - Sequential reads within a block chunk can proceed at high speed - similar to continuous allocation
- How do you find an available block group?
  - Use freelist bitmap to find block of 0's.

4/3/06

Joseph CS162 ©UCB Spring 2006

Lec 18.18

### Large File Version of DEMOS



- What if need much bigger files?
  - If need more than 10 groups, set flag in header: BIGFILE
    - » Each table entry now points to an indirect block group
  - Suppose 1000 blocks in a block group ⊃ 80GB max file
    - » Assuming 8KB blocks, 8byte entries ⊃  
(10 ptrs \* 1024 groups/ptr \* 1000 blocks/group) \* 8K = 80GB
- Discussion of DEMOS scheme
  - Pros: Fast sequential access, Free areas merge simply  
Easy to find free block groups (when disk not full)
  - Cons: Disk full ⊃ No long runs of blocks (fragmentation), so high overhead allocation/access
  - Full disk ⊃ worst of 4.1BSD (lots of seeks) with worst of continuous allocation (lots of recompaction needed)

4/3/06

Joseph CS162 ©UCB Spring 2006

Lec 18.19

### How to keep DEMOS performing well?

- In many systems, disks are always full
  - CS department growth: 300 GB to 1TB in a year
    - » That's 2GB/day! (Now at 3-4 TB!)
  - How to fix? Announce that disk space is getting low, so please delete files!
    - » Don't really work: people try to store their data faster
  - Sidebar: Perhaps we are getting out of this mode with new disks... However, let's assume disks full for now
- Solution:
  - Don't let disks get completely full: reserve portion
    - » Free count = # blocks free in bitmap
    - » Scheme: Don't allocate data if count < reserve
  - How much reserve do you need?
    - » In practice, 10% seems like enough
  - Tradeoff: pay for more disk, get contiguous allocation
    - » Since seeks so expensive for performance, this is a very good tradeoff

4/3/06

Joseph CS162 ©UCB Spring 2006

Lec 18.20

## UNIX BSD 4.2

- Same as BSD 4.2 (same file header and triply indirect blocks), except incorporated ideas from DEMOS:
  - Uses bitmap allocation in place of freelist
  - Attempt to allocate files contiguously
  - 10% reserved disk space
  - Skip-sector positioning (mentioned next slide)
- Problem: When create a file, don't know how big it will become (in UNIX, most writes are by appending)
  - How much contiguous space do you allocate for a file?
  - In Demos, power of 2 growth: once it grows past 1MB, allocate 2MB, etc
  - In BSD 4.2, just find some range of free blocks
    - » Put each new file at the front of different range
    - » To expand a file, you first try successive blocks in bitmap, then choose new range of blocks
  - Also in BSD 4.2: start files from same directory near each other

4/3/06

Joseph CS162 ©UCB Spring 2006

Lec 18.21

## Attack of the Rotational Delay

- Problem 2: Missing blocks due to rotational delay
  - Issue: Read one block, do processing, and read next block. In meantime, disk has continued turning: missed next block! Need 1 revolution/block!



- Solution1: Skip sector positioning ("interleaving:")
  - » Place the blocks from one file on every other block of a track: give time for processing to overlap rotation
- Solution2: Read ahead: read next block right after first, even if application hasn't asked for it yet.
  - » This can be done either by OS (read ahead)
  - » By disk itself (track buffers). Many disk controllers have internal RAM that allows them to read a complete track
- Important Aside: Modern disks+controllers do many complex things "under the covers"
  - Track buffers, elevator algorithms, bad block filtering

4/3/06

Joseph CS162 ©UCB Spring 2006

Lec 18.22

## How do we actually access files?

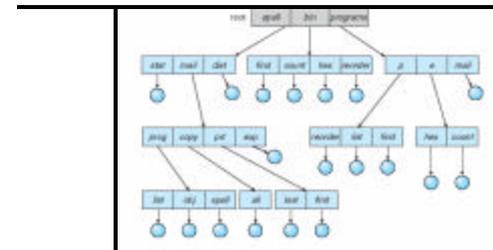
- All information about a file contained in its file header
  - UNIX calls this an "inode"
    - » Inodes are global resources identified by index ("inumber")
  - Once you load the header structure, all the other blocks of the file are locatable
- Question: how does the user ask for a particular file?
  - One option: user specifies an inode by a number (index).
    - » Imagine: open("14553344")
  - Better option: specify by textual name
    - » Have to map name® inumber
  - Another option: Icon
    - » This is how Apple made its money. Graphical user interfaces. Point to a file and click.
- Naming: The process by which a system translates from user-visible names to system resources
  - In the case of files, need to translate from strings (textual names) or icons to inumbers/inodes
  - For global file systems, data may be spread over globe. Need to translate from strings or icons to some combination of physical server location and number

4/3/06

Joseph CS162 ©UCB Spring 2006

Lec 18.23

## Directories



- Hierarchical name space: Files named by ordered set (e.g.: /programs/p/list)
- Directories: a special type of relation
  - Just a table of (file name, inumber) pairs
  - Question: how is the relation stored?
    - » Directories often stored just like files
    - » Can store inumber for directories or files in other directories
  - Question: how is the directory structured?
    - » Needs to be quickly searchable!

4/3/06

Joseph CS162 ©UCB Spring 2006

Lec 18.24

### Where are inodes stored?

- In early UNIX and DOS/Windows' FAT file system, headers stored in special array in outermost cylinders
  - Header not stored anywhere near the data blocks. To read a small file, seek to get header, see back to data.
  - Fixed size, set when disk is formatted. At formatting time, a fixed number of inodes were created (They were each given a unique number, called an "inumber")
- Later versions of UNIX moved the header information to be closer to the data blocks
  - Often, inode for file stored in same "cylinder group" as parent directory of the file (makes an *ls* of that directory run fast).
  - Pros:
    - » Reliability: whatever happens to the disk, you can find all of the files (even if directories might be disconnected)
    - » UNIX BSD 4.2 puts a portion of the file header array on each cylinder. For small directories, can fit all data, file headers, etc in same cylinder → no seeks!
    - » File headers much smaller than whole block (a few hundred bytes), so multiple headers fetched from disk at same time

4/3/06

Joseph CS162 ©UCB Spring 2006

Lec 18.25

### Summary

- File System:
  - Transforms blocks into Files and Directories
  - Optimize for access and usage patterns
  - Maximize sequential access, allow efficient random access
- File (and directory) defined by header
  - Called "inode" with index called "inumber"
- Multilevel Indexed Scheme
  - Inode contains file info, direct pointers to blocks,
  - indirect blocks, doubly indirect, etc..
- DEMOS:
  - CRAY-1 scheme like segmentation
  - Emphized contiguous allocation of blocks, but allowed to use non-contiguous allocation when necessary
- Naming: the process of turning user-visible names into resources (such as files)

4/3/06

Joseph CS162 ©UCB Spring 2006

Lec 18.26