Disk Allocation

ECE 695

Nov 1

Y. Charlie Hu

1

[lec20] Why Files?

- Physical reality
  - Block oriented
  - Physical sector numbers
  - No protection among users of the system
  - Data might be corrupted if machine crashes
- File system abstraction
  - Byte oriented
  - Named files
  - Users protected from each other
  - Robust to machine failures

"I will save our lab3 solution on platter 5, track 8739, sector 3-4."

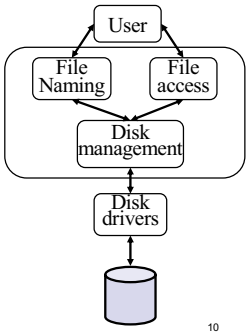
"My lab3 solution is in ~ece595/lab3/memory.c"

3

3

Roadmap

- Functionality (API)
  - Basic file system
    - Data structures / disk layout
    - File operations
  - Directories
- Performance
  - → Disk allocation
  - Disk scheduling
  - Buffer cache



10

10

Disk Allocation Problem

- Definition: allocate disk blocks when a file is created or grows, and free them when a file is removed or shrinks
- Does this sound familiar?
- How are they different?
  - Granularity?
  - Access performance?
- Shall we approach it like segmentation or paging?

11

11

Disk Allocation Problem

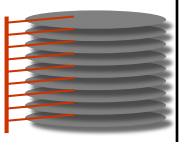
- Two tasks:
  - How to allocate blocks on disk for a file?
  - How to design inode to keep track of blocks?
- Why are they hard?
  1. Disk performance characteristics
  2. Uncertainty about file sizes and access patterns

13

13

Disk Mechanics & Performance

- Platter / Head / Tracks / Sectors / Cylinders
- Rotation 1000's of RPM (7200, 10k, 15k)
- Avg seek 5-10 ms



- Assume
  - 255 heads *38913 tracks * 63 sectors * 512 bytes = 320GB
  - Seek time =6ms, 7200 RPM ➔ rotational latency = 8ms
- Block access time = seek time + rotational latency + reading time
  - Accessing a random block: 6ms + 4 ms + 8ms/63 = 638ms/63
  - Accessing the block right after: 8ms/63
- Implications?

14

14

Disk vs. Memory

Memory	Disk
• Latency in 100's of processor cycles	• Latency in milliseconds • 1ms = 10^6 cycles on 1Ghz machine
• Transfer rate ~ 1000 MB/s (DDR SDRAM)	• Transfer rate in 30KB/s -- 30MB/s

• Contiguous allocation gains ~10x
  • Cache hits
  • RAS/CAS (DRAM)
	• Contiguous allocation gains ~100x

15

15

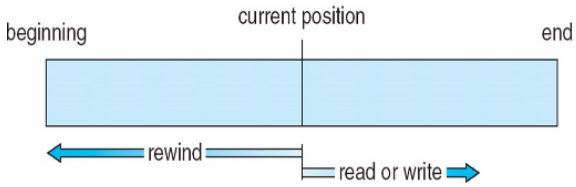
Uncertainty about Files Sizes & Usage Patterns

- File usage determines locality -- How do users access files?
  - Sequential: bytes read in order
  - Random: read/write element out of middle of arrays
  - Whole file or partial file
- How do file sizes vary? (determines inode design)
  - Most files are small
  - Large files use up most of the disk space
  - Large files account for most of the bytes transferred
- Bad news
  - Want FS to be efficient for all cases

16

16

Sequential File Access



17

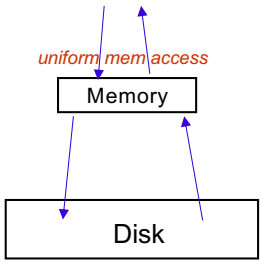
Demand Paging vs. Disk I/O

18

18

[lec15] Demand Paging algorithms

- Optimal
- FIFO
- FIFO with 2nd chance
- Clock: a simple FIFO with 2nd chance
- Enhanced FIFO with 2nd chance NFU
- Approximate LRU



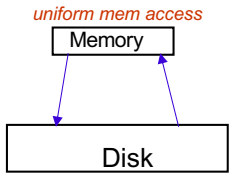
- Definition: pick victim page to swap to disk upon page fault
- Focus on reducing number of page faults (going to disk)
  - Don't care which page fault, don't care where on disk
- What kind of *locality* matters?

19

19

Disk Allocation Problem

- Definition: allocate disk blocks when a file is created or grows, and free them when a file is removed or shrinks



read(fd, buf, size)

20

- Cannot do anything about number of I/O -- Focus on performance of I/Os (that go to disk), i.e., where on disk
  - What kind of locality matters?

20

Design goals and expectation

- Optimize I/O performance
  - What can we do for random access patterns?
  - What can we do for sequential access patterns?
- Also want to minimize file header size
  - File header: for keeping track of blocks
  - Ideally fit in inode

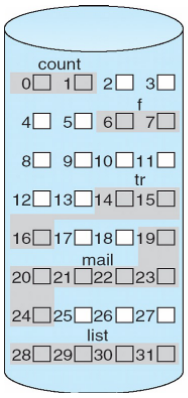
Disk Allocation Methods

- Contiguous
- Single-level indexed (Cray-1)
- Linked
- FAT (MS-DOS, OS/2)
- Multi-level indexed (UNIX)

Contiguous Allocation

- Request in advance for the size of the file
- Search bit map or linked list of free blocks to locate a space
- File header contains
  - first sector number
  - number of sectors

Contiguous Allocation of Disk Space



directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Analogy in memory management?

Contiguous Allocation

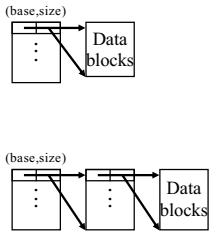
- Request in advance for the size of the file
- Search bit map or linked list to locate a space
- File header contains
  - first sector number
  - number of sectors
- Pros
  - Fast sequential access
  - Easy random access
- Cons
  - External fragmentation
  - Hard to grow files

Extent-Based Systems

- Later file systems (i.e. Veritas File System – 1st commercial journal FS) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in **extents**
- An **extent** is a contiguous block of disks
  - Extents are allocated for file allocation
  - A file consists of one or more extents

Example: DEMOS (OS for Cray-1, 1977)

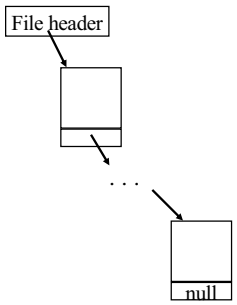
- Approach
  - Using contiguous allocation
  - Allow non-contiguous
- Approach
  - 10 (base,size) pointers
  - Indirect for big files
- Pros & cons
  - Can grow (max 10GB)
  - fragmentation
  - Difficult to grow each segment



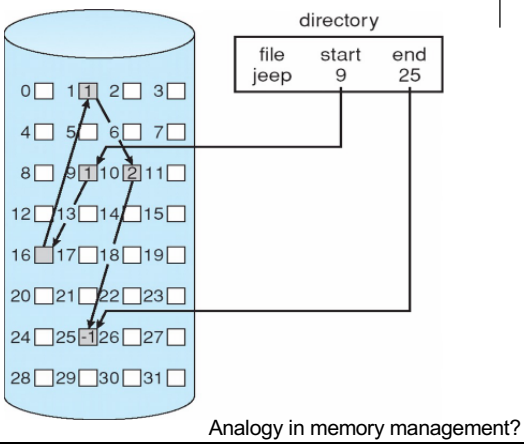
Analogy in memory management?

Linked Files

- Approach
  - File header points to 1st block on disk
  - Each block points to next
- Pros
  - Can grow files dynamically
  - No external fragmentation
  - Sequential access easy
- Cons
  - random access: horrible
  - unreliable: losing a block means losing the rest



Linked Allocation



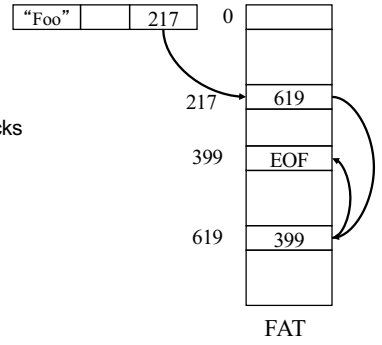
Analogy in memory management?

29

A variation of linked files: File Allocation Table (FAT)

(MS-DOS, OS2 by Microsoft / IBM)

- Approach
  - A section of disk for each partition is reserved
  - One entry for each block
  - A file is a linked list of blocks
  - A directory entry points to the 1st block of the file
- Pros
  - Pros of linked alloc.
  - Simpler than linked alloc.
- Cons
  - Always go to FAT
  - Random access slow for large files

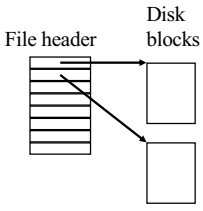


30

30

Single-Level Indexed Files

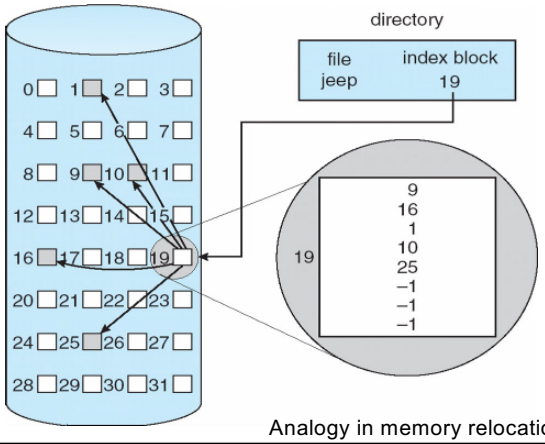
- A user declares max size
- A file header holds an array of pointers to point to disk blocks
- Pros
  - Can grow up to a limit
  - Random access is fast
  - No external fragmentation
- Cons
  - Clumsy to grow beyond limit
  - Up-front declaration a real pain
  - File header cannot fit in inode
    - large inodes would be wasteful



31

31

Example of Indexed Allocation

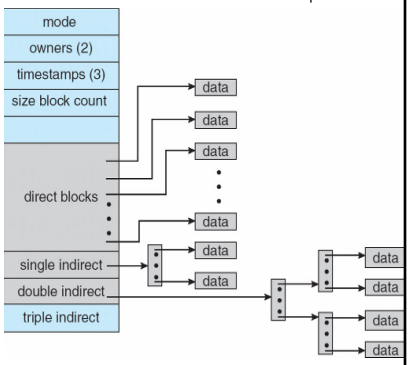


Analogy in memory relocation?

32

Multi-Level Indexed Files (UNIX)

- 13 Pointers in a header
  - 10 direct pointers
  - 11: 1-level indirect
  - 12: 2-level indirect
  - 13: 3-level indirect
- Pros & Cons
  - In favor of small files
  - Can grow
  - Limit is 16G (w/ 1K block)
  - Random access has up to 4 seeks
- How many disk accesses to load block 23, 5, 340?



Analogy in memory management?

33

Linux ext2

- From Linux kernel documentation for ext2:

  "*There are pointers to the first 12 blocks which contain the file's data in the inode. There is a pointer to an indirect block (which contains pointers to the next set of blocks), a pointer to a doubly indirect block and a pointer to a trebly indirect block.*"

- ext2 has 15 pointers.
  - Pointers 1 to 12 point to direct blocks
  - pointer 13 points to an indirect block
  - pointer 14 points to a doubly indirect block
  - pointer 15 points to a trebly indirect block

34

34

Deep thinking

- Is multi-level indexing a good idea for page table design?

36

36

Summary

- Seeks kill performance → exploit spatial locality
- Extent-based allocation optimizes sequential access
- Single-level indexed allocation has speed
- Unix file system has great flexibility
- Bitmaps show contiguous free space
- Linked lists easy to search for free blocks

37

37

Reading

- Chapters 10-11

38