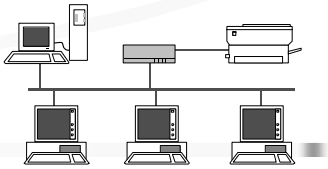


Operating Systems 600.418

File System (cont.) and Disk Management

Department of Computer Science
The Johns Hopkins University

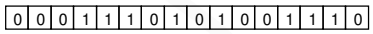
File System (cont.) and Disk Management

Lecture 8

Reading:
Silberschatz & Galvin
chapter 11,13
Additional Reading:
Stallings
chapter 12, 11

Free Space Management

- Bit map.
  - **Every bit represents a block on the disk.**
  - **1: the block is free. 0: the block is allocated.**
  - **Therefore, we spend 1 block of bits for every 4096*8 = 32768 blocks.**
  - **Stored on a specific place on the disk.**
  - **Has to be kept also in memory to be efficient.**
  - **Special hardware support (bit operation) to find the first '1' in a word if it is not zero.**
  - **Efficient in finding consecutive blocks.**

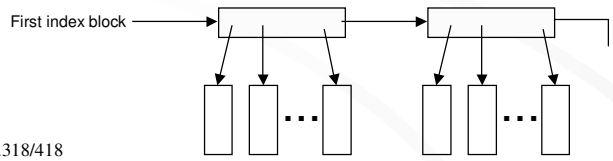


Free Space Management (cont.)

- Link List.
  - **Keeping a pointer to the first free block in a special location on the disk.**
  - **Each free block contains a pointer to the next free block.**
  - **Allocating one block is relatively ok.**
  - **Allocating several blocks is expensive (disk movements).**
- FAT (MS/DOS, OS/2):
  - **Improved variation of the link list.**
  - **In the FAT file system method, the free space is managed as one file.**

Free Space Management (cont.)

- Grouping. For example:
  - **4Kbytes size blocks, 4bytes block pointers.**
  - **Each index block contains 1023 pointers to free blocks**
  - **Each index block also contains a pointer to the next index block.**
  - **Therefore, we spend 1 block of index for every 1023 free blocks.**
  - **Fast allocation of large number of free blocks.**

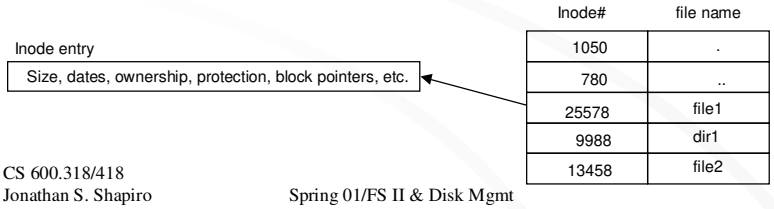


Free Space Management (cont.)

- Counting.
  - **Taking advantage of the fact that several contiguous blocks may be allocated or freed simultaneously.**
  - **Every entry in the free list contains the address of the first free block and the number of consecutive free blocks.**
  - **Fast allocation of large number of consecutive free blocks.**

Directory Implementations

- Unix
  - Directories are implemented as a type of file.
  - Each directory entry contains a file name and an inode (index node) number.
  - The rest of the information associated with the file is located in the inode section of the volume.
  - Every directory has an entry representing itself (.) and another entry representing its parent (..)
  - Root inode is known.



Directory Implementations

- Unix (cont.)
  - Each volume contains:
    - Boot block
    - Super block - file system parameters such as size of the file system (number of blocks), free blocks list, size of inode section, etc.
    - inode section.
    - Data space.
  - Accessing the file file_name
    - If /a/b/c, the root inode is known. If a/b/c, the current directory inode is know. Start from there as working inode.
    - While not end of path
      - check if working inode is a directory, check permission
      - locate name in directory and get inode of that name.
    - return working inode.

Directory Implementations

- MS/DOS
  - **The root directory place is known and its size is constant.**
  - **Every other directory is managed as a file.**
  - **Each entry in the directory has the following fields.**
    - File name + extension.
    - Attributes
    - date + time
    - size
    - first block number - from here we can use the FAT to access random blocks in the file.
  - **Each directory entry is of 32 bytes.**

File System Consistency

- Lots of effort goes into these disks, that are, after all, just large data structures.
- In most operating systems, the file system on the disk can be in inconsistent state as a result of a power failure or system crash.
- Many time, this can be resolved (maybe for a price).
- FSCK procedure in Unix:
  - **Compare Blocks in files and in the free list. Every block has to be accounted for exactly once. If not:**
    - Does not appear even once - link to free list.
    - Appears in the free list and in a file - extract from free list.
    - Appears more than once in files - very bad - replace one of them with a copy and notify.

File System Consistency (cont.)

- FSCK process in Unix (cont.):
  - **Compare Files and inodes. The count in the inode needs to be equal to the number of directory entries that point to it. If not:**
    - If count in inode is smaller - increase count and we might have a problem.
    - If count in inode is larger - decrease count.
  - **More checks can be added in principle.**
- Some file systems operate similarly to database systems (with logs etc.) so that they will not enter such cases. What is the cost?

Backup

- Well, disks might crash sometimes
- We can protect data by backing it up to another storage device.
- Tape is a very cost effective method to store data that does not need to be accessed online.
- A typical backup schedule:
  - Day 1: Complete backup
  - Day 2: Increment from day 1.
  - Day 3: Increment from day 2.
  - …
  - Day *n*: Increment from day *n*-1.
  - Go back to day 1.

Block (Buffer) Cache

- Once blocks are read from the disk to main memory, the blocks maybe cached there.
- What kind of block replacement algorithms can be used here?
- FIFO, second chance, etc. can all be used. But actually in this case, it is practical to use strict LRU. Why?
- However, Strict LRU might not be the best. Can you think about some reasons? Can you think of useful modifications?

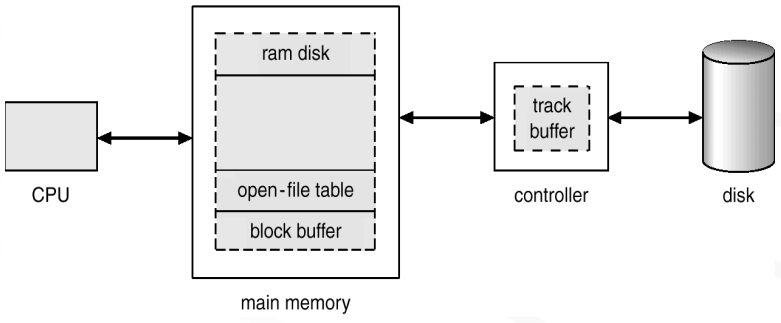
Block Cache (cont.)

- Modified LRU that takes into account:
  - Will we need these blocks again ?
    - Maybe we closed the file.
    - Free-behind - maybe it is a sequential operation.
  - Is this block crucial for consistency (for example - it contains the inode) ?
    - We might need to write it to disk immediately. We can still keep it in memory, of course.
    - MS/DOS method - write through cache.
  - Read-ahead - we can read, together with the requested block, the following blocks from the disk and cache them.
  - It might be useful to sync the dirty blocks to disk from time to time.

RAM Disk

- A section of main memory is set aside and treated as a virtual disk.
- The RAM disk device driver accepts all standard disk operations but performs them in memory - very fast.
- Will not survive power failure or reboot.
- Common in PC systems.
- What is the difference between RAM disk and buffer cache?

Various memory caches to support the disk system

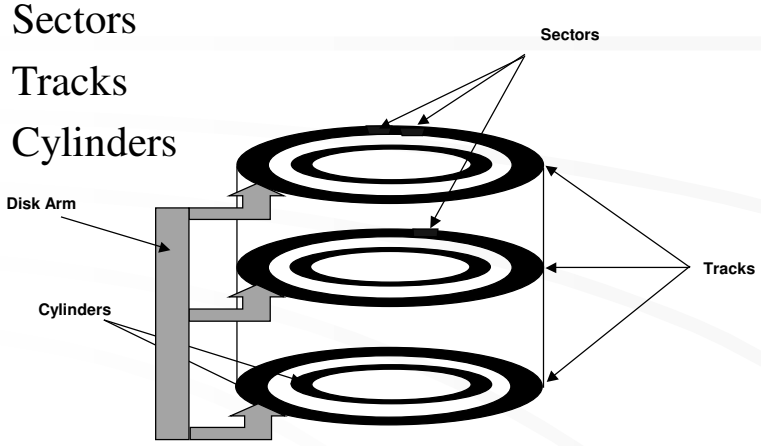


Other Performance Issues

- Block allocation:
  - **It pays to allocate blocks near existing blocks for the same file.**
  - **To implement this, a bit map free space system is best.**
- Inode location:
  - **Put the inode section in the middle of the disk.**
- Interleaving
  - **The next block is stored on the disk further on the same cylinder so that when the program ask for the next block, the head is not-yet there.**
- Directory as a hash table instead of a list.

Disk Structure

- Sectors
- Tracks
- Cylinders



A disk is accessed as an array: sector 0 is the first sector of the top track of the outmost cylinder. The next sectors of the same track are then ordered. Then, sectors from the next track are ordered. After all the sectors of all the tracks are ordered, we move to the next cylinder. The innermost is the last cylinder.

Disk Performance Parameters

- Seek time
  - **The time it takes to move the disk arm to the required cylinder.**
  - **In old disk - proportional to the distance.**
  - **Newer disks - more complicated - accelerates and then decelerates.**
- Rotational latency
  - **The time it takes for the disk to rotate so that the required sector is above the disk head.**
  - **On average, half a rotation is required to get in position.**

Disk Performance Parameters

- Access time.
  - **The time it takes to get in position to read or write.**
  - **Seek time + rotational latency.**
- Data Transfer occurs as the sectors move under the head,so the bandwidth is determined by the rotation rate of the disk.
- Time to get the data:
  - **Access time + transfer time.**
- The disk system has memory that might allow it to cache the whole track.

Disk Scheduling

- A disk operation request specifies several pieces of information:
  - **Is this input or output ?**
  - **Disk address for the transfer.**
  - **Memory address for the transfer.**
  - **Number of bytes to be transferred.**

- When a request is initiated by a process, if the disk is available, the request can be serviced immediately.
- If the disk is busy, a request queue can form.
- Upon completion of a disk request, the operating system can choose which pending request to service next.
- We discuss several disk scheduling algorithms that determine how to choose the next request to service.

Disk Scheduling (Cont.)

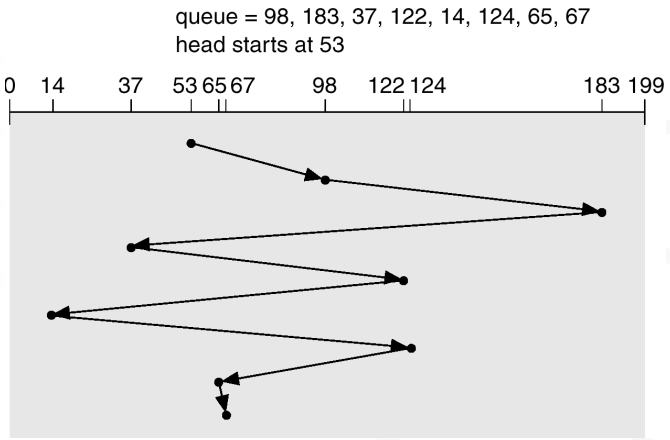
- Effective bandwidth is determined by the disk parameters, the specific scenario, and the disk scheduling algorithm used.
- We illustrate the different algorithms assuming a disk with 200 cylinders (0-199) and the following scenario.

98, 183, 37, 122, 14, 124, 65, 67

- We assume that in the beginning, the disk head is at cylinder 53.

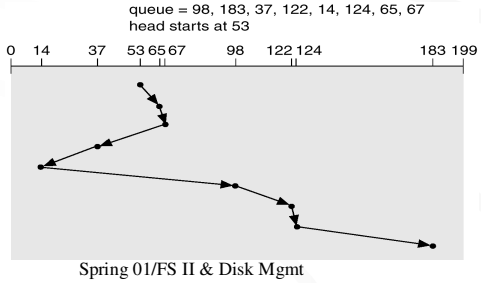
First Come First Served (FCFS)

- Total head movement of 640 cylinders.



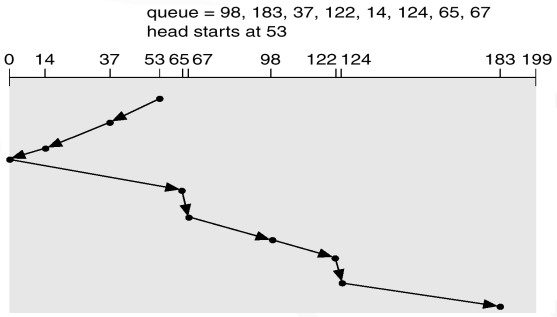
Shortest Seek time First (SSTF)

- Selects the request with the minimum seek time from the current head position.
- SSTF scheduling is a form of Short Job First scheduling. May cause starvation of some requests.
- Is it optimal ?
- Total head movement of 236 cylinders. (208 is possible - can you think how?)



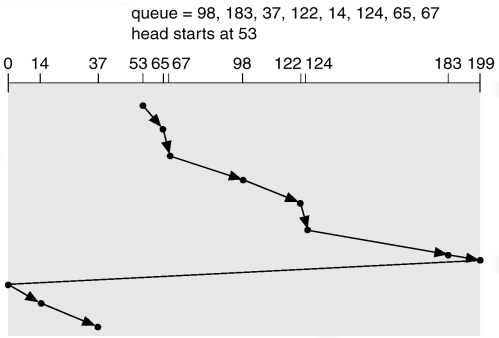
SCAN

- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.
- Sometimes called the *elevator algorithm*.
- Total head movement of 236 cylinders.



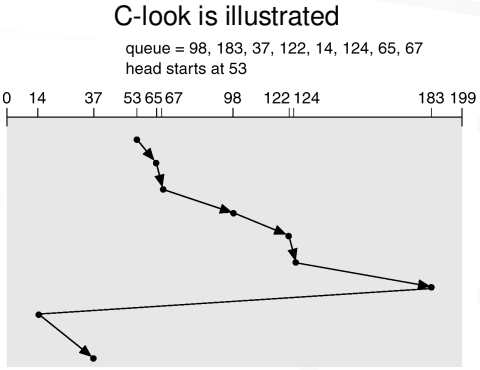
C-SCAN

- The head moves from one end of the disk to the other. servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.
- Tradeoffs compared with SCAN?
- Provides a more uniform wait time than SCAN.



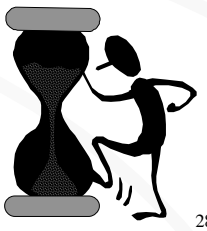
LOOK, C-LOOK

- Version of SCAN, C-SCAN
- Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.



N-step-SCAN

- Segments the request queue to subqueues with length of N, and service them according to SCAN.
- Why?
- So we are more fair - not stuck only on one side of the disk if requests keep coming.
- What do we get if N=1 ?
- If N=1 we have FCFS.
- Large N -> SCAN.



FSCAN

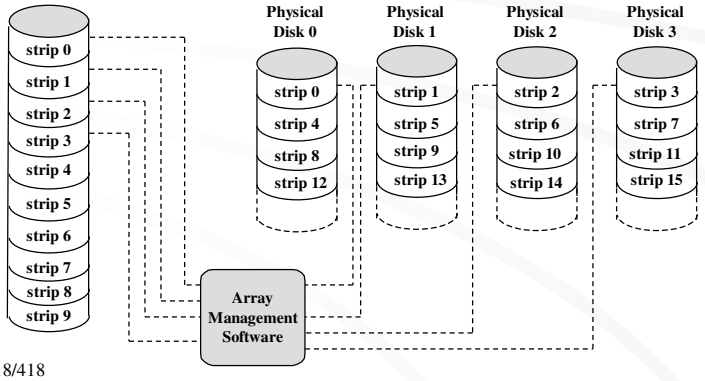
- Two Queues.
- While treating a queue, incoming requests enter the other queue.
- When all the requests in the queue are serviced, switch to the other queue.
- Treat each queue using SCAN.

The RAID Concept

- Redundant Array of Independent Disks (some say Inexpensive Disks).
- A set of physical disk drives viewed by the operating system as a single logical drive.
- Data is distributed across the physical drives of an array.
- Redundant disk capacity is used to store parity information, which guarantees data recoverability in case of a disk failure.

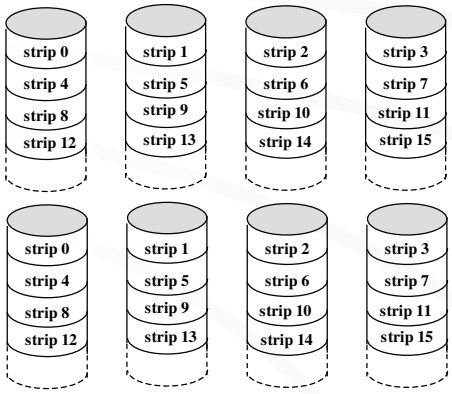
RAID Level 0

- No redundancy.
- Great performance:
  - **small strips - high data transfer capacity (parallel disk).**
  - **Large strips - high I/O request rates (parallel queues).**



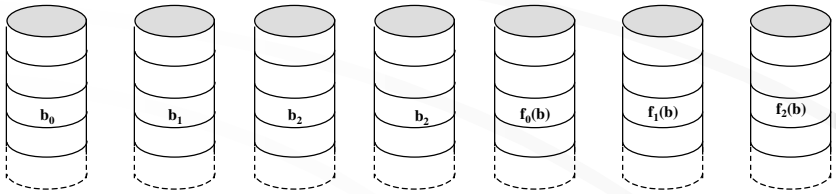
RAID Level 1 (1+0)

- Mirroring.
- A read request can be served by either disk. The one with the arm in the better position is selected.
- A write request requires both disks to write.



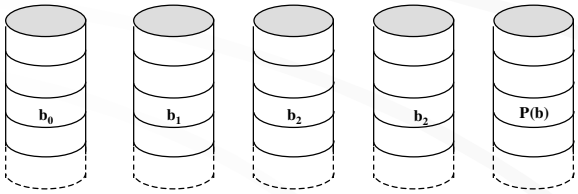
RAID Level 2

- Redundancy using Hamming error correcting code.
- Each strip is very small - one byte or word.
- Read and write requests require all the disks.
- Given high reliability of disks, RAID Level 2 is an overkill and is never implemented.



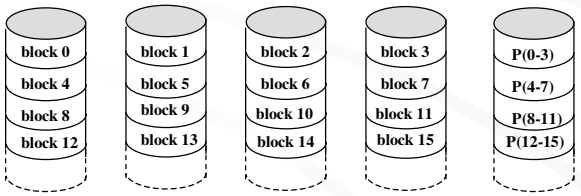
RAID Level 3

- Redundancy using parity bit.
- Each strip is very small - one byte or word.
- Read and write requests require all the disks.
- Great transfer rate, poor I/O request rate.
- Recovery from failure is simple (XOR).



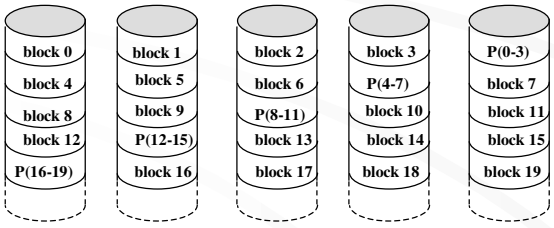
RAID Level 4

- Block level parity.
- Read requires the relevant disk.
- Write requires the relevant disk and the parity disk.
- Recovery from failure is simple (XOR).
- High I/O request.
- Can you see a performance problem?



RAID Level 5

- Block level interleaved parity.
- Read requires the relevant disk.
- Write requires the relevant disk and the parity disk **for the particular block**.
- Recovery from failure is simple (XOR).



Homework

A computer system has a 4GBytes disk and 128MBytes of memory with the following parameters:
- Register access time: negligible (**zero**).
- Memory access time: 1 micro second
- Disk access time: 10 mili second to get/put the data (including transfer time).

The system is a virtual memory system with demand paging. The operating system manages the page table in direct access.
When we access a virtual address, if the required page is not in memory, the operating system brings the page from the disk and restarts the operation.
There is a 20% likelihood that the page that is brought into memory replaces a dirty page that has to be written to the disk.

Assume that the process page table is completely in memory when the process is running.

1. Assuming that the page fault rate is 99.999% (the likelihood of a virtual memory access to be a "hit"). What is the system effective access time to memory?

2. How can we improve this virtual memory system by adding a TLB using a small array of registers? What will be the best (lower bound) effective access time to memory that can be achieved in this way?

3. Going back to the system in question 1. We add a 256KBytes cache to the CPU with access time of 100 nano second. Assume that the cache key is a virtual address.
   What the cache hit ration needs to be in order for the effective access time to memory to be 500 nano seconds.

Homework (cont.)

Today, it is not realistic to maintain the complete page table in memory.

4, 5, 6. Redo 1, 2, 3, assuming that the page table is managed with regular pages that can have page faults (as opposed to direct access). Assume that the page faults rate and the dirty-page likelihood for page table pages is similar to that of regular pages.

318 students: do only questions 1, 2, 3. Each of which is 33 points.

418 students: do questions 1-6. Each of which is 16.5 points.

One point to clearly type your name and the course number :)

PLEASE EXPLAIN EVERY STEP OF YOUR CALCULATIONS!!!!

Please type your submission.

Please submit by Tuesday, November 21 in class.