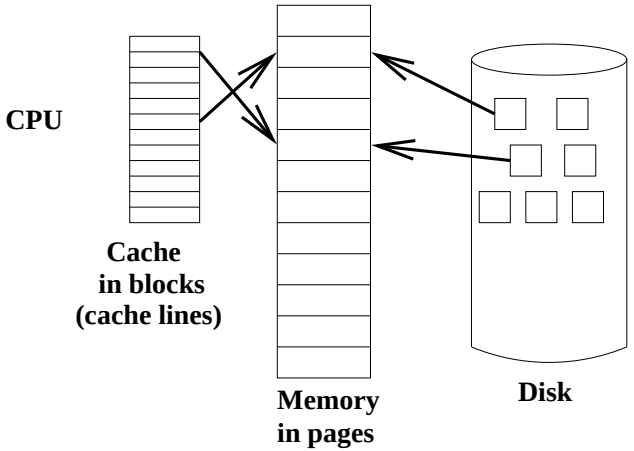
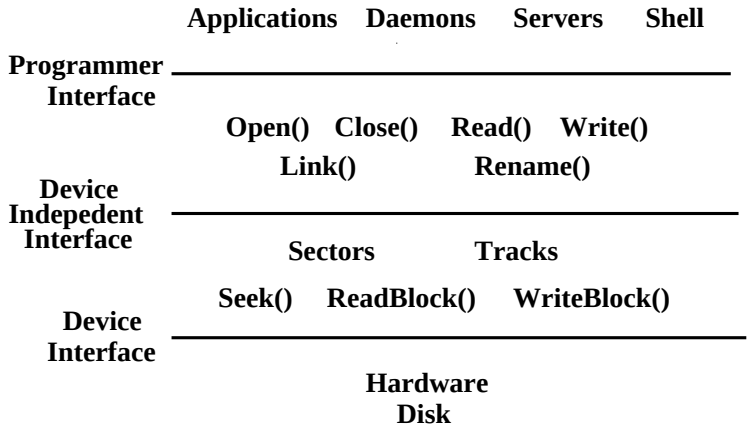
Memory and Architecture: The Big Picture



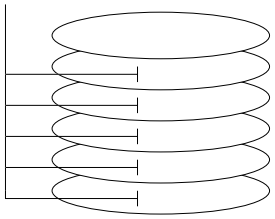
Remember the high-level view of the OS as a translator from the user abstraction to the hardware reality.

User Abstraction		Hardware Resource
Processes/Threads		CPU
Address Space	$\Longleftarrow$ **OS** $\Longrightarrow$	Memory
Files		Disk

File System Abstraction

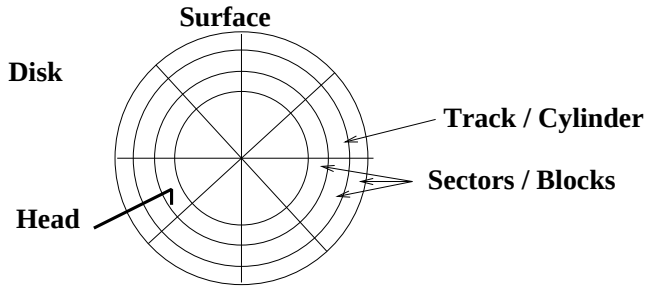


How Disks Work



- CDs come individually, but disks come organized in *disk pack* consisting of a stack of platters. Disk packs use both sides of the platters, except on the ends.
- Comb has 2 read/write head assemblies at the end of each arm.
- *Cylinders* are matching sectors on each surface.
- Disk operations are in terms of radial coordinates.
  1. Move arm to correct track, waiting for the disk to rotate under the head.
  2. Select and transfer the correct sector as it spins by.

How Disks Work Today



- The disk surface is circular and is coated with a magnetic material. The disk is always spinning (like a CD).
- Tracks are concentric rings on disk with bits laid out serially on tracks.
- Each track is split into *sectors* or *blocks*, the minimum unit of transfer from the disk.

To read or write a disk block:

Overhead: time the CPU takes to start a disk operation

Seek: (latency) position head over a track/cylinder. The seek time depends on how fast the hardware moves the arm.

Rotational delay: (latency) time for the sector to rotate underneath the head. Rotational delay depends upon how fast the disk spins.

Transfer time: (bandwidth) time to move the bytes from the disk to memory.

I/O Time = overhead + seek + rotational delay + transfer

Typical Disk Parameters

	Good PC disk	High-end disk
Disk Capacity	4 GB	36.4 GB
platters per pack	10	10
tracks per surface	3832	11540
sectors per track	134	215
bytes per sector	512	732
revolutions per minute	7200	7200
average seek time	8.5 ms	7.5 ms
average rotational latency	4.17 ms	4.17 ms
buffer to host burst transfer rate	20 MB/sec	200 MB/sec
buffer size	1 MB	4 MB
size	3.5 inches	3.5 inches

Access Time

Key: to get the quickest disk response time, we must minimize seek time and rotational latency:

- Make disks smaller
- Spin disks faster
- Schedule disk operations to minimize head movement
- Lay out data on disk so that related data are on nearby tracks.
- Place commonly-used files where on the disk?
- We should also pick our sector size carefully:
  - If the sector size is too small, we will have a low transfer rate because we will need to perform more seeks for the same amount of data.
  - If our sector size is too large, we will have lots of internal fragmentation.

Disk Head Scheduling

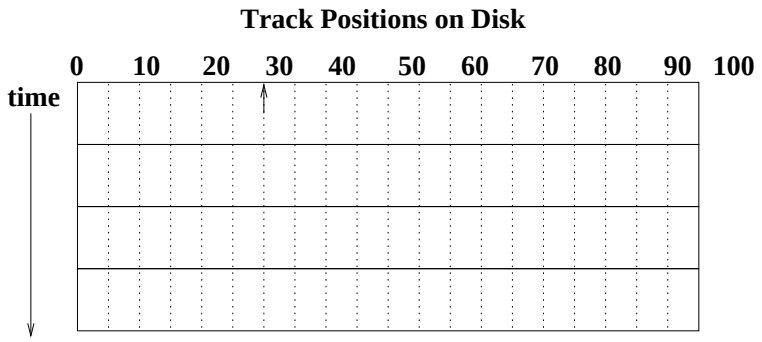
Idea: Permute the order of disk requests from the order that they arrive from the users to an order that reduces the length and number of seeks.

1. First-come, first-served (FCFS)
2. Shortest seek time first (SSTF)
3. SCAN algorithm (0 to 100, 100 to 0, 0 to 100, ...). If there is no request between current position and the extreme (0 or N), we don't have to seek there.
4. C-SCAN circular scan algorithm (0 to 100, 0 to 100, ...)

FCFS Disk Head Scheduling

Example requests: 65, 40, 18, 78

1. FCFS - service the requests in the order that they come in

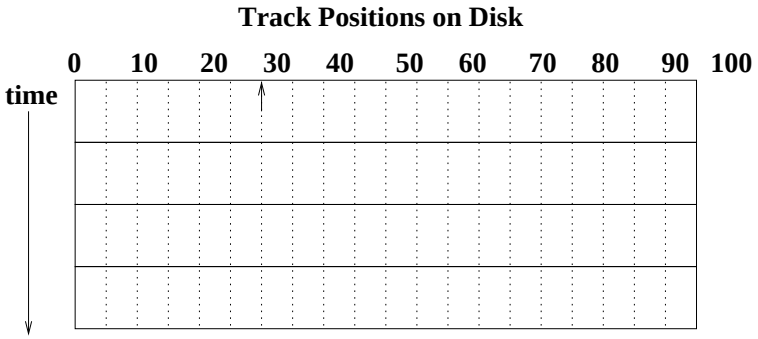


- Order of seeks:
- Distance of seeks:

- When would you expect this algorithm to work well?

Disk Head Scheduling

2. SSTF - always go to the next closest request

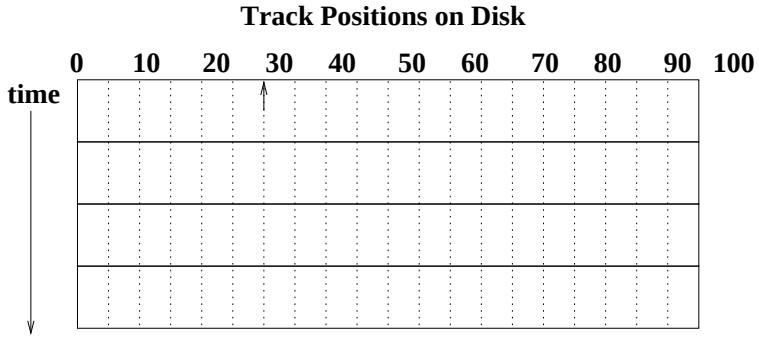


- Order of seeks:
- Distance of seeks:

- Can implement this approach by keeping a doubly linked sorted list of requests.
- Is this efficient enough?
- Is it optimal?
- Problems?

Disk Head Scheduling

3. SCAN - head moves back and forth across the disk (0 to 100, 100 to 0, 0 to 100, ...), servicing requests as it passes them

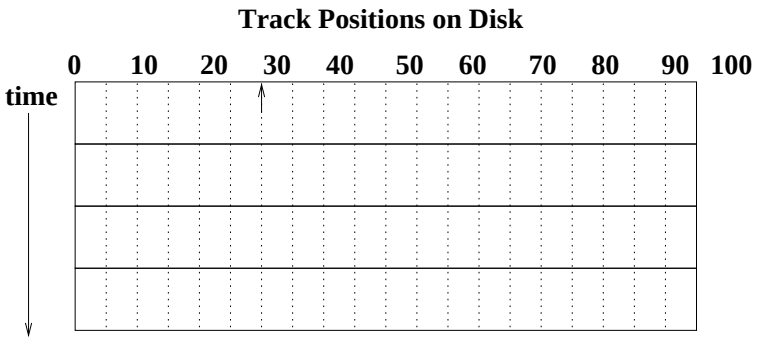


- Order of seeks, assuming the head is currently moving to lower numbered blocks:
- Distance of seeks:

- Requires a sorted list of requests.
- Simple optimization does not go all the way to the edge of the disk each time, but just as far as the last request.

Disk Head Scheduling

4. C-SCAN - circular scan algorithm (0 to 100, 0 to 100, ...)



- Order of seeks:
- Distance of seeks:

- More uniform wait times for requests. Why?

Other Ways to Improve Disk Performance

Disk Interleaving

Problem: Contiguous allocation of files on disk blocks only makes sense if the OS can react to one disk response and issue the next disk command before the disk spins past the next block.

Idea: Interleaving - Don't allocate blocks that are physically contiguous, but those that are temporally contiguous relative to the speed with which a second disk request can be received and the rotational speed of the disk. Might use every second or third block.

Read Ahead

Idea: read blocks from the disk ahead of user's request and place in buffer on disk controller.

Goal: reduce the number of seeks - read blocks that will probably be used while you have them under the head.

We considered pre-fetching virtual pages into physical memory, but decided that was difficult to do well since the future is difficult to predict. Is disk read-ahead any better?

Summary

- Disks are slow devices relative to CPUs.
- For most OS features, we are very concerned about efficiency.
- For I/O systems, and disk, in particular, it is worthwhile to complicate and slow down the OS if we can gain improvement in I/O times.

Review Questions:

- What property of disks can we use to make the insertion, deletion, and access to the lists of requests fast?
- Rank the algorithms according to their expected seek time.
- Is SCAN or SSTF fairer?
- Is SCAN or C-SCAN fairer?