Announcements/Reminders

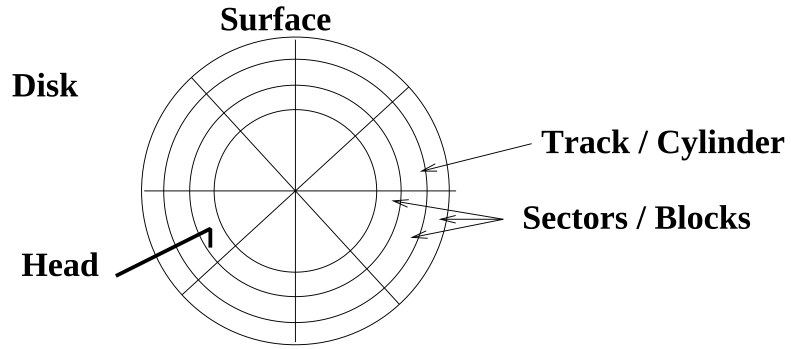
- HW 6 due today (Friday)
- Lab 4 due on Monday (please turn in on time!)

Last Class: I/O Systems

- I/O is expensive for several reasons:
  - Slow devices and slow communication links
  - Contention from multiple processes.
  - I/O is typically supported via system calls and interrupt handling, which are slow.

- Approaches to improving performance:
  - Reduce data copying by caching in memory
  - Reduce interrupt frequency by using large data transfers
  - Offload computation from the main CPU by using DMA controllers.
  - Increase the number of devices to reduce contention for a single device and thereby improve CPU utilization.
  - Increase physical memory to reduce amount of time paging and thereby improve CPU utilization.

Today: Secondary Storage



- **Recall: to read or write a disk block:**
  - **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
  - **Disk I/O Time** = seek + rotational delay + transfer.

Typical Disk Parameters (2 years out of date)

	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 the 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

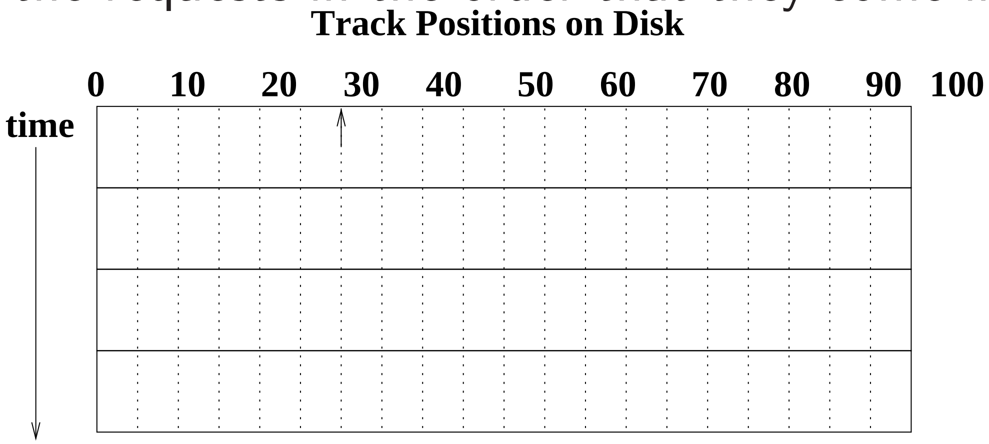
- Typically several disk read requests have been made (by one process, by several processes, or by the OS)
- **Idea:** Permute the order of disk requests from the order that they arrive from the users to an order that reduces the length and number of seeks.

  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

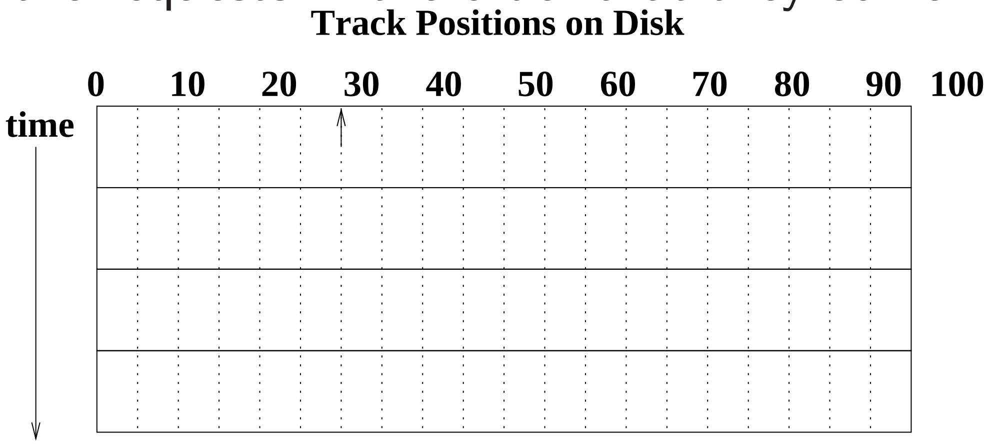


- Order of seeks:
- Distance of seeks:
- When would you expect this algorithm to work well?

FCFS Disk Head Scheduling

Example requests: 65, 40, 18, 78

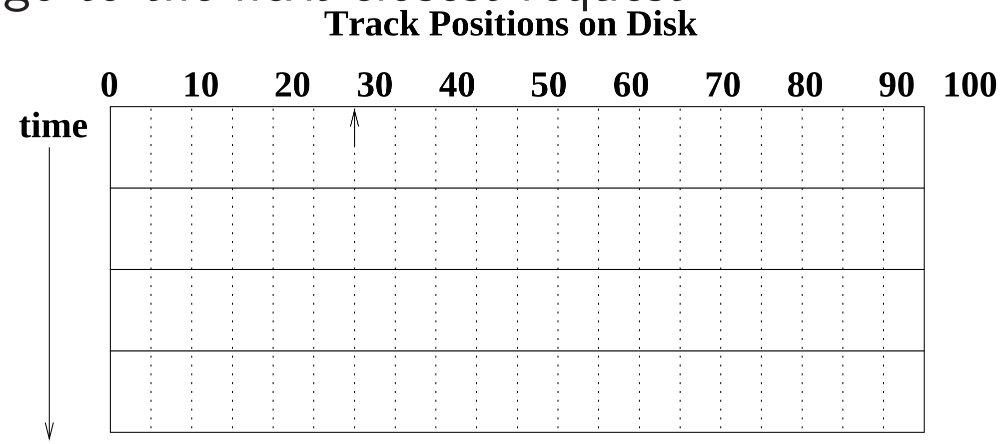
1. FCFS - service the requests in the order that they come in



- Order of seeks: 65, 40, 18, 78
- Distance of seeks: $35 + 25 + 22 + 60 = 142$
- When would you expect this algorithm to work well?
  If the requests were near one-another.

SSTF Disk Head Scheduling

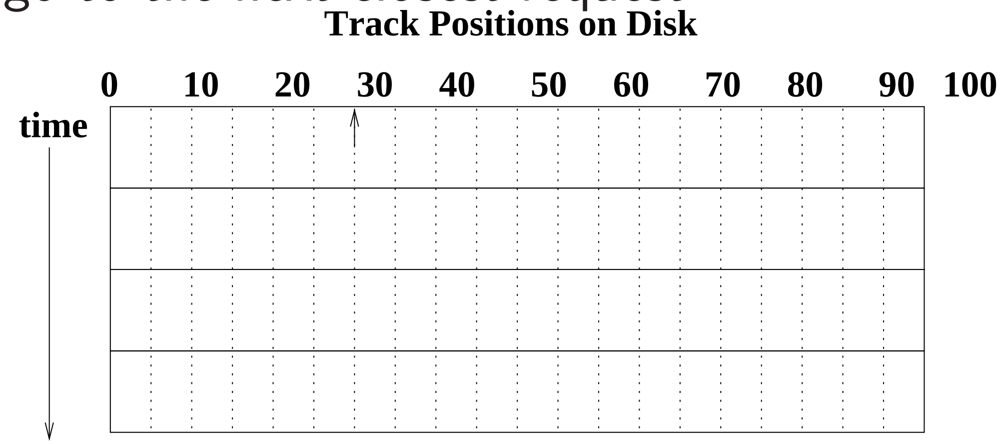
- SSTF: always go to the next closest request



  - 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?

SSTF Disk Head Scheduling

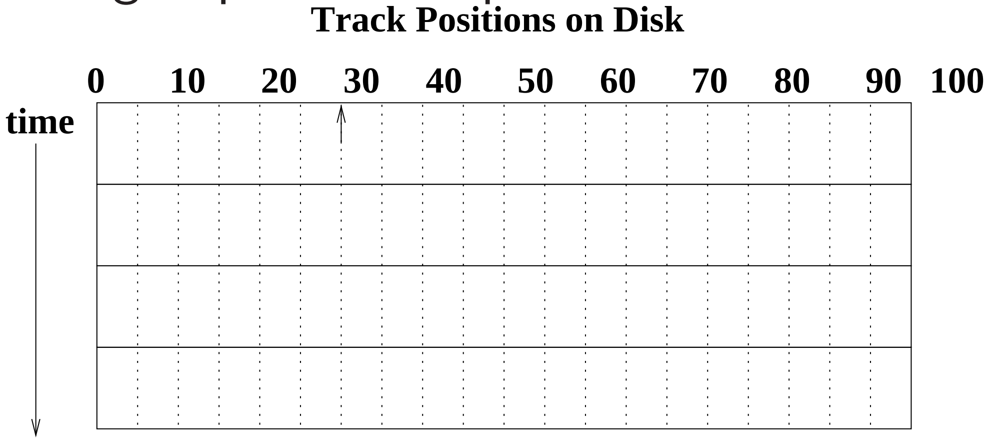
- SSTF: always go to the next closest request



  - Order of seeks: 40, 18, 65, 78
  - Distance of seeks: $10 + 22 + 47 + 13 = 92$
  - Is this efficient enough? Yes, the cost of computation is minor relative to the seek time.
  - Is it optimal? No, it's greedy. Minimizes cost of next seek, but not overall cost.
  - Problems? Potential for starvation

SCAN Disk Head Scheduling

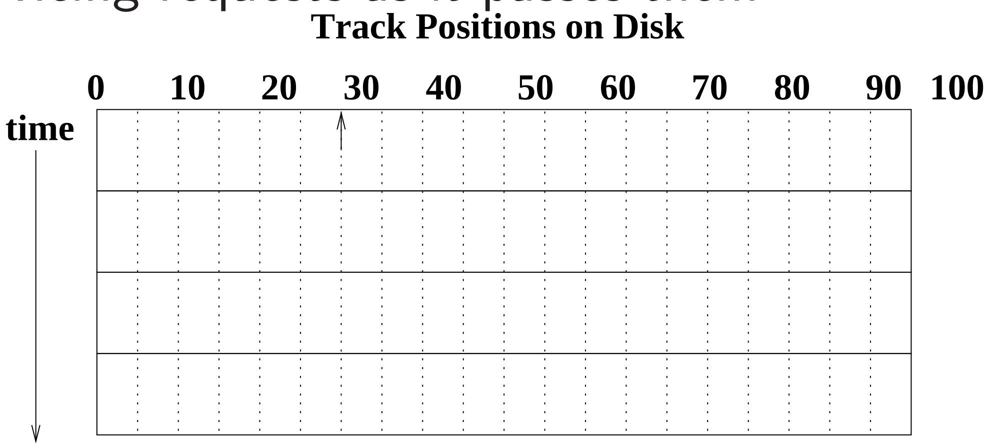
- SCAN: head moves back and forth across the disk (0 to 100, 100 to 0, 0 to 100, ...), servicing requests as it passes them



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

SCAN Disk Head Scheduling

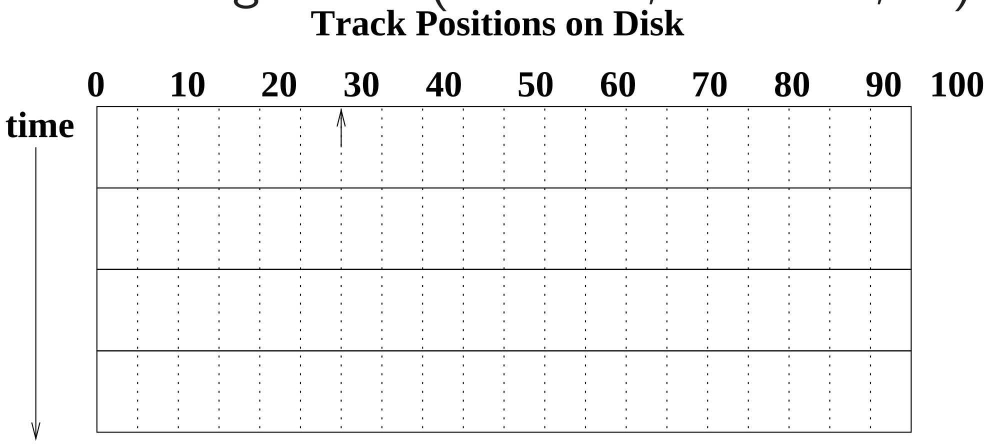
- SCAN: head moves back and forth across the disk (0 to 100, 100 to 0, 0 to 100, ...), servicing requests as it passes them



  - Order of seeks, assuming the head is currently moving to lower numbered blocks: 18, 40, 65, 78
  - Distance of seeks: $12 + 22 + 25 + 13 = 72$
  - Requires a sorted list of requests.
  - Simple optimization: do not go all the way to the edge of the disk each time, but just as far as the last request.

C-SCAN Disk Head Scheduling

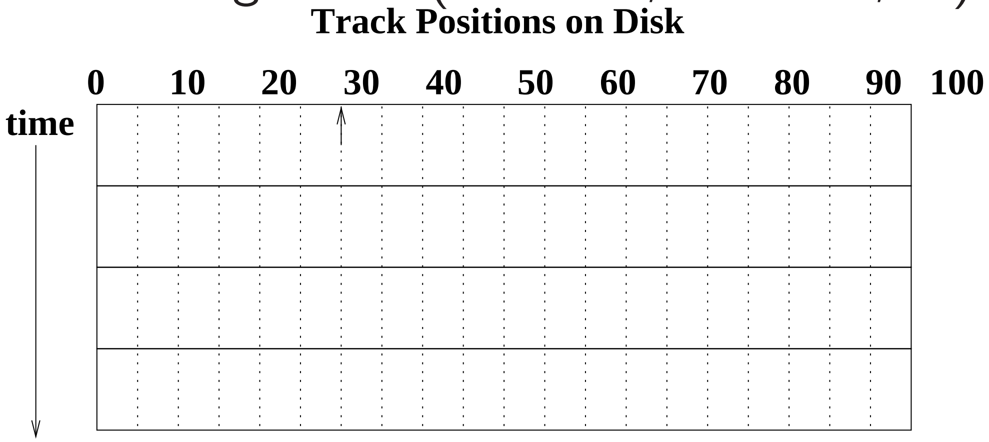
- C-SCAN: circular scan algorithm (0 to 100, 0 to 100, ...)



  - Order of seeks:
  - Distance of seeks:

C-SCAN Disk Head Scheduling

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



  - Order of seeks: 40, 65, 78, 18
  - Distance of seeks: $10 + 25 + 13 + 60 = 108$
  - More uniform wait times for requests. Why? At the end of a pass, most of the requests are at the other end. With SCAN, a request might need to wait for 2 almost-complete sweeps of the disk before it would get processed.

Improving Disk Performance using 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.

Improving Disk Performance using 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?

Improving Disk Performance using Read Ahead

Yes, for sequentially accessed files.

Tertiary Storage

- Lower cost devices than secondary storage (disks)
- Typically Slower, BUT: larger and cheaper than disks
- Used primarily for storing archival data or backups.
  - tape drives
  - Jazz and Zip drives
  - Optical disks: Write once read-many (WORM), CD-R, CD-RW
  - Robotic jukeboxes
- Primary, secondary and tertiary devices form a storage hierarchy

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?

Summary

- **Review Questions:**
  - What property of disks can we use to make the insertion, deletion, and access to the lists of requests fast?
    Disk operations are very slow relative to the CPU/memory operations required to maintain these lists.
  - Rank the algorithms according to their expected seek time.
    C-SCAN $>$ SCAN $>$ SSTF (*** dependent on the query set)
  - Is SCAN or SSTF fairer?
    SCAN
  - Is SCAN or C-SCAN fairer?
    C-SCAN

Next Time

- Distributed File Systems (Chapter 17)