EECS 482
Introduction to Operating Systems

Winter 2018

Harsha V. Madhyastha

Recap: CPU Scheduling

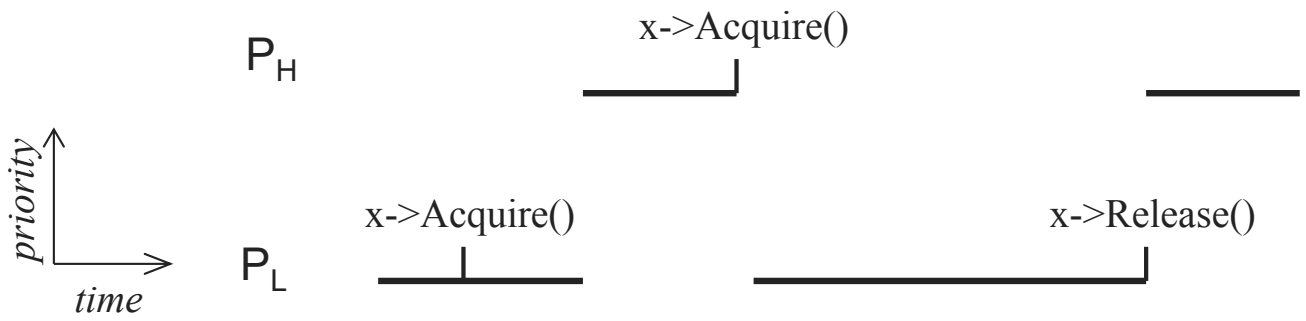
- First Come First Serve (FCFS)
  - Simple, but long waiting times for short jobs
- Round Robin
  - Reduces waiting times for short jobs
  - Higher context switching overhead
- Shortest Time to Completion First (STCF)
  - Optimal average response time
  - Use runtime of a job so far to predict remaining time

Priority

- Priority
  - Assign external priority to each job
  - Run high-priority jobs before low-priority ones
  - Use, e.g., round-robin for jobs of equal priority
  - Prone to starvation

- Methods for preventing starvation?
  - If job has not run for time t, boost priority
  - Handle priority inversion (lock held by low-priority)

Priority Inversion



Multimedia: Soft real-time

- Examples:
  - Audio should not skip when compiling projects
  - Predictable: video player plays n frames per sec

- Can reserve a share of the CPU
  - X% of the CPU over some time interval
  - Unused CPU split among remaining jobs

Hard real-time scheduling

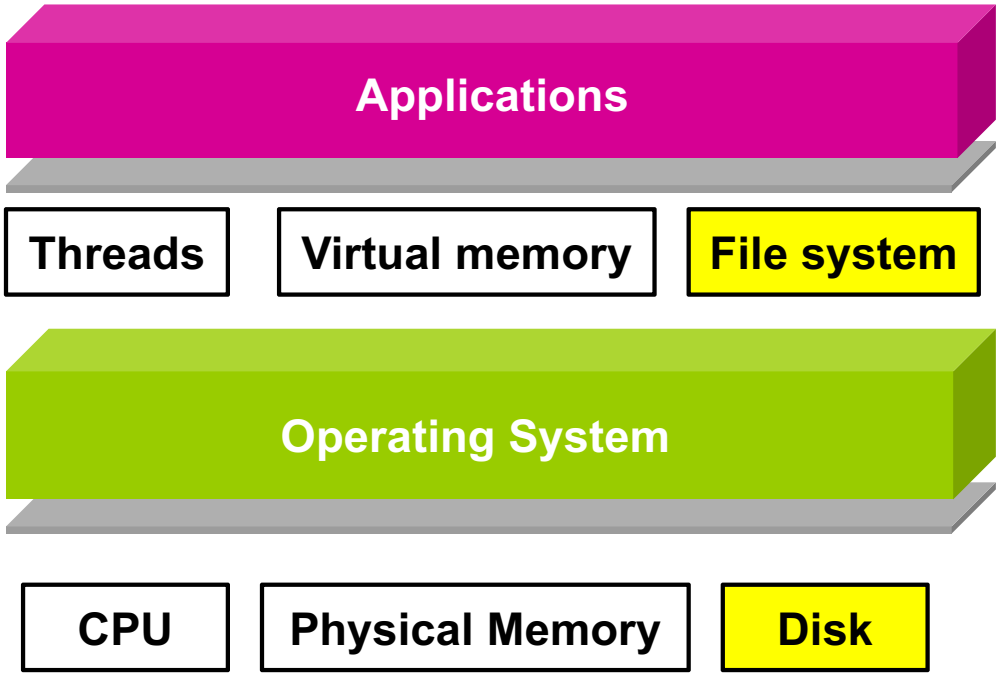
- Jobs have to complete before deadline
  - Demand / deadline known in advance
  - Example: vehicle control, aviation, etc.

- Earliest-deadline first (EDF)
  - Always run jobs whose deadline is soonest
  - Preempt if newly arriving job has earlier deadline
  - Always succeeds if schedule is feasible
  - But, may be very poor if schedule is infeasible

Scheduling: Summary

- Many different policies
  - FCFS
  - Round robin
  - STCF
  - Priority
  - Proportional share
  - EDF
- Scheduling strategy in grocery stores?
- OS schedulers mix all of these
  - Many heuristics and complex tuning

OS Abstractions



- Next few lectures:
  - What interface does file system export to apps?
  - How does file system interact with hardware?

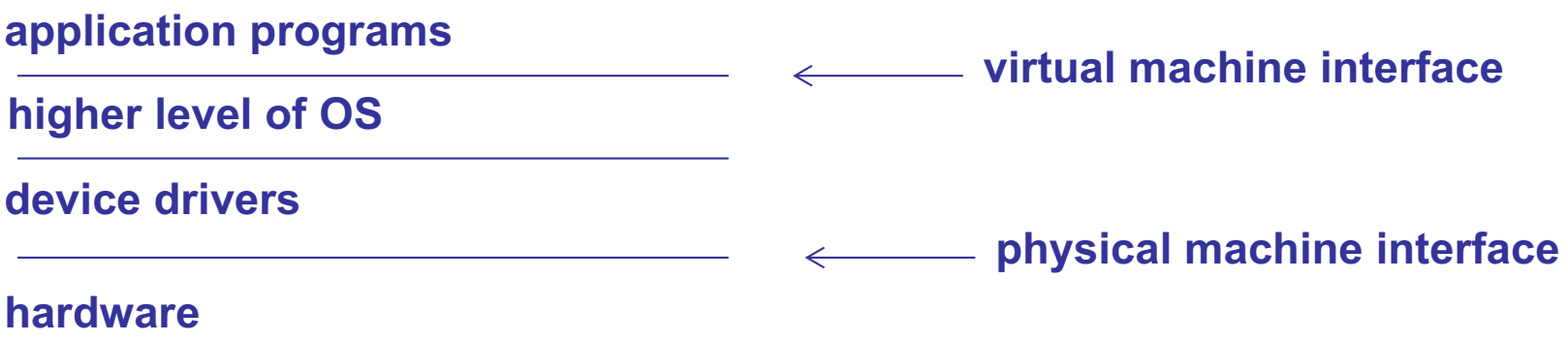
Reality vs. Abstraction

- Hardware interface
  - Small set of disks, with array of blocks on each disk
  - Interface varies across disks
  - Slow, and potentially inconsistent on crash

- OS abstraction
  - Large set of files, with rich naming convention
  - Same interface to files, irrespective of hardware
  - Fast, and crash consistent

Dealing with heterogeneity

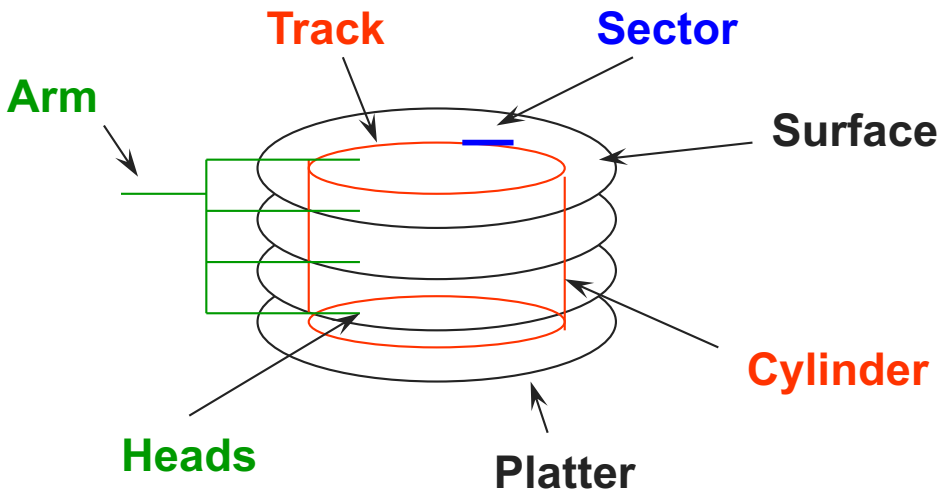
- Problem: Wide range of disk types and interfaces
  - How to manage this diversity?
- Solution: Add device driver abstraction inside OS

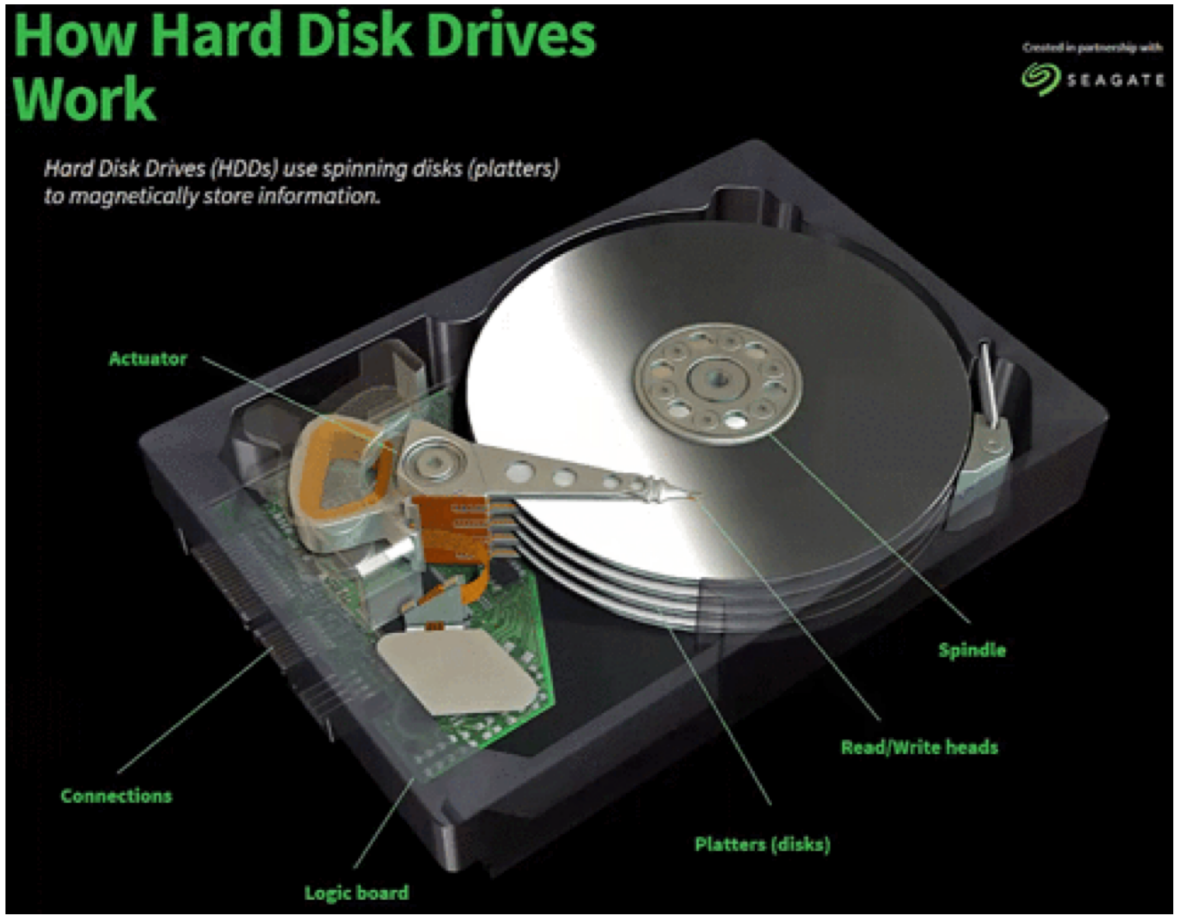


  - Hide differences among different brands and interfaces
  - Minimize differences between similar types of devices

Physical Hard Disk Structure

- Disk components
  - Platters
  - Surfaces
  - Tracks
  - Sectors
  - Cylinders
  - Arm
  - Heads




How Hard Disk Drives Work
Created in partnership with
SEAGATE
Hard Disk Drives (HDDs) use spinning disks (platters) to magnetically store information.
Actuator
Spindle
Read/Write heads
Connections
Platters (disks)
Logic board

Hard Disk Performance

- What does disk performance depend upon?
  - Queue – wait for the disk to be free
  - Positioning – move the disk arm to the correct cylinder and rotate to the right sector
  - Access – transfer data from/to disk

- For given load, performance depends on
  - Positioning overhead (~1-10 ms)
  - Transfer time (~100 MBps)

Disks Heterogeneity

- Seagate Barracuda 3.5" (workstation)
  - capacity: 250 - 750 GB
  - rotational speed: 7,200 RPM
  - sequential read performance: 78 MB/s (outer) - 44 MB/s (inner)
  - seek time (average): 8.1 ms
- Seagate Cheetah 3.5" (server)
  - capacity: 73 - 300 GB
  - rotational speed: 15,000 RPM
  - sequential read performance: 135 MB/s (outer) - 82 MB/s (inner)
  - seek time (average): 3.8 ms
- Seagate Savvio 2.5" (smaller form factor)
  - capacity: 73 GB
  - rotational speed: 10,000 RPM
  - sequential read performance: 62 MB/s (outer) - 42 MB/s (inner)
  - seek time (average): 4.3 ms

Optimizing I/O performance

- To increase performance of slow I/O devices:
  - Avoid doing I/O (Disks are slooooow!)
  - Reduce overhead (minimize positioning time)
  - Amortize overhead over larger requests

- Efficiency = transfer time / (positioning time + transfer time)
  - Rule of thumb: Achieve at least 50% efficiency
  - Example: 5 ms avg. seek time and 100MBps transfer rate → Read at least 500KB

Project 3

- Write test case for every transition in your state machine
- Even without handling fork(), test with multiple processes to test vm_destroy
  - Run in multiple terminals or in background
  - Call fork() before any calls to vm_map
- filename argument to vm_map is user-level virtual address

Disk scheduling

- Reduce overhead by reordering requests
  - Can be implemented in OS or hardware (Tradeoffs?)
- Pick 1 of n requests in queue:
  - Example: 98, 183, 37, 122, 14, 124, 65, 67
  - Start track is 53

- FCFS (first come, first served)
  - 98, 183, 37, 122, 14, 124, 65, 67
  - Total head movement: 640 tracks

SSTF

- Pick 1 of n requests in queue:
  - Example: 98, 183, 37, 122, 14, 124, 65, 67
  - Start track is 53

- SSTF (shortest seek time first)
  - 65, 67, 37, 14, 98, 122, 124, 183
  - Total head movement: 236 tracks
- Any drawbacks?
  - Potential starvation

SCAN

- Pick 1 of n requests in queue:
  - Example: 98, 183, 37, 122, 14, 124, 65, 67
  - Start track is 53

- SCAN (like windshield wipers)
  - 37, 14, 65, 67, 98, 122, 124, 183
  - Total head movement: 208 tracks
- Drawbacks and fix?
  - Blocks in the middle served more often than at ends
  - C-SCAN: Serve requests only in one direction

SCAN vs. STCF

- SCAN typically has better throughput
  - Minimizing total head movement
- SSTF may have better response time
  - Servicing fastest request first
  - But, poor throughput can cause longer queue waits
- Disk scheduling: Queueing vs. positioning
- Does CPU scheduling affect throughput?

Anticipatory scheduling

- Consider two processes with disk locality
  - P1: read 1, compute, read 2, compute, read 3, ...
  - P2: read 1001, compute, read 1002, compute, ...
- What behavior will SSTF give?
  - 1, 1001, 2, 1002, 3, ... (suboptimal)
- Key idea: wait for a bit for new request
  - 1, 2, 3, (timer expires), 1001, 1002
  - Can improve throughput **and** response time

Optimizing data layout

- Keep related items together on disk

- What items will be accessed together?
  - Can guess based on general usage patterns
    - » Blocks in same file often accessed together
    - » Files in same directory often accessed together
    - » Files often accessed with their directory
  - Can guess based on past accesses of data
    - » Learn patterns and reorganize data on disk

Flash (solid state disks)

- Optimizations depend on specifics of a device
- Flash differs from magnetic disk
  - Better random read performance
  - Lower positioning overhead
  - Starting to yield more read parallelism
  - Lower power
  - Better shock resistance
  - But also wearout, cannot overwrite data in place
- OS hides physical characteristics of device from applications

Optimizing for Flash

- Move data blocks to do wear leveling
- Write data in big blocks
- Asynchronously erase blocks
- Prefer to read data rather than write

Next time ...

- File system interface and structure