May the Force be with you … and adieu
The power of abstractions —layered
each layer hiding a tangled truth beneath

Processes fence off one another —self-contained,
while threads run concurrent lives in a shared computing space

The scheduler decides
who speaks next in the chorus of tasks
caution, however: deadlocks lurk around the resource corner,
where no task yields, and all is frozen

Memory management, the illusionist, stretches what is finite,
beyond the silicon bounds of our beloved RAM
powered by paging and segmentation
a sleight of hand that makes bloatware possible

The file system arranges blocks into meaning,
weaving together files as a tapestry of blocks

What seems a breeding ground for chaos,
now dances in harmony
and hums with activity
The OS built on unseen scaffolds – that we *now* know

CS 370: Operating Systems [Disk Scheduling Algorithms]

Shrideep Pallickara
Computer Science
Colorado State University

COLORADO STATE UNIVERSITY

1

Frequently asked questions from the previous class survey

- Flash memory
  - More susceptible to hardware errors? Why do we set it to a logical "1"?
  - When you have lower capacity, and make modifications constantly, does this wear out the cells faster?
  - Do we need defragmentation here?
  - Is the slack space provided by manufacturers available for general use?
- RAIDs: Do we have nesting of these levels?
- What's your favorite sci-fi movie?

2

Topics covered in this lecture

- Coping with system failures
- Swap space management
- Disk scheduling algorithms
- Final Exam

3

4

Coping with system failures

- File system structures are maintained on disk and in memory
- Operations result in *structural changes* to the file system on disk
  - Changes may be interrupted by a crash
- System failures should not result in
  - **Data Loss**
  - **Inconsistencies** among data structures

Sources of inconsistency

- OSes **cache** to optimize performance
  - If cached changes do not reach disk?
    - Corruption
- **Bugs** may also corrupt a file system
  - File system implementation
  - Disk controllers
  - Applications

Inconsistency example: File creation

- Directory structure is modified, inode is set aside, etc
- Free inode count may indicate that an inode has been allocated
  - But the directory structure may not point to it

Consistency checking: Approaches

(I) Scan all metadata of file system
- Confirm or deny consistency
- Time consuming

(II) Record state within file system metadata
- At start of metadata change the *status bit* set
  - Metadata is in flux
- If metadata updates complete successfully
  - Clear the status bit
- If bit is set: a **consistency checker** is run

Consistency checker compares structure with data on disk and tries to fix inconsistencies

- Allocation & free space management algorithms dictate efficiency and success
- Linked list allocation
  - Link exists from block to block
  - File can be recreated
- Indexed allocation
  - Loss of inode entry is disastrous
    - File blocks have **no knowledge** of each other

Some issues with consistency checking

- Inconsistency may be **irreparable**
  - Inability to recover structures
  - Loss of files (possibly entire directories)
- Can require **human intervention** for conflict resolution
  - Unavailable until this is performed
- Can be very **time consuming**
  - Can take up to several hours

11

Applying log-based recovery techniques to file system METADATA UPDATES

- All metadata changes are written sequentially to a **log**
- Changes written to log are considered **committed**
  - System call can return
- Log entries are **replayed** across actual file system structures

12

Some things about the log file

- Implemented as a circular buffer
  - When action is completed
    - Buffer entry is removed and pointer is advanced
- Log **location**
  - Separate section of the file system
  - Perhaps on a separate disk
    - Efficiency

After a system crash the log is inspected

- Log will contain zero or more transactions
- If there are **non-zero** transactions
  - Not completed but committed by the OS
    - Must be completed
  - Aborted transaction: Not committed before crash
    - Undone
- Recovery is much more **targeted**

Benefits of using logging for disk metadata updates

- Costly synchronous random metadata writes
  - **Become** (less expensive) synchronous, sequential writes to the logging area
- Changes in the log are **replayed asynchronously** to appropriate disk structures
  - Random writes
- Updates are **much faster** than when they are applied directly to on-disk structures

Other approaches

- **Never overwrite** blocks with new data {Used in Journaling}
- All data and metadata changes in new blocks
- When transaction completes
  - Structures **updated to point** to the new block
- Old blocks can be reused
  - If NOT, a snapshot preserves view before the update
- ZFS **checksums** all data and metadata blocks

SWAP SPACE MANAGEMENT

Virtual memory uses the disk space as an extension of main memory

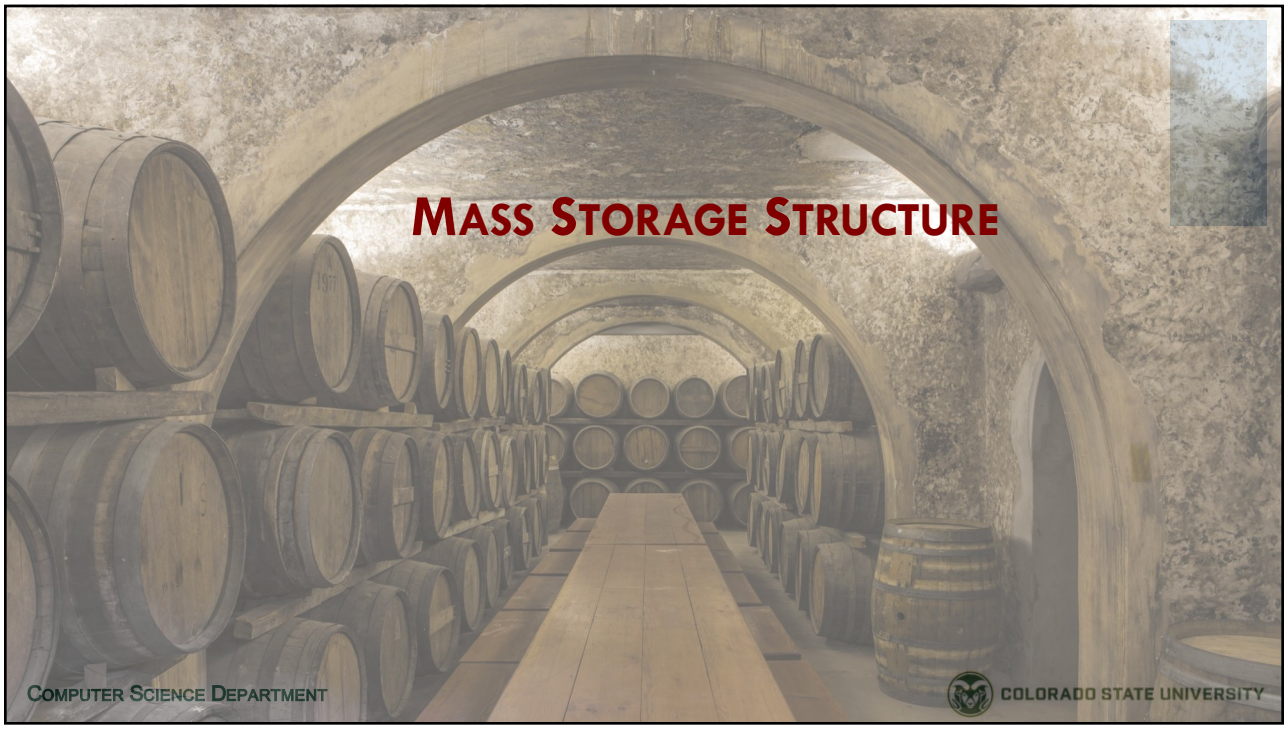
- Using swap space **decreases** system performance
- Main objective of swap space
  - Provide best possible **throughput** for the virtual memory system

Swap space location

- Not in the purview of the normal file system
  - Navigating directory and allocation data structures
    - Time consuming
    - Could result in additional disk accesses
- Use a **raw** partition
  - Separate swap-space manager
  - (De)allocate blocks from the raw partition

Using the raw partition

- Swap space *accessed more frequently* than the file system
- Algorithms are optimized for **speed** *not efficiency*
- Internal fragmentation may be higher
  - BUT swap space data have shorter life spans
    - Acceptable trade-off



21

Disk drives are attached to the computer by a set of wires called an I/O bus

- Enhanced integrated drive electronics (EIDE)
- Advanced technology attachment (ATA)
- Serial ATA (SATA)
- Universal serial bus (USB)
- Small computer systems interface (SCSI)
- NonVolatile Memory express (NVMe)

22

Disk controllers are built into each disk drive

- Computer **places** a command using memory mapped I/O ports
- Disk controller **operates** the disk hardware to perform command

Magnetic tape

- Early secondary-storage medium
- **Slow access** times
  - Moving to the correct spot on tape is time consuming
  - 1000 times slower than magnetic disk for random access
- Mainly for **backup**

Disk Structure

- One dimensional **array** of blocks
  - Mapped onto sectors of the disk
- Sector **0**
  - 1st sector of 1st track on outermost cylinder
    - A cylinder holds a group of tracks
- Mapping proceeds from the outermost cylinders to the innermost one

Sectors on a hard disk

- {*cylinder, track, sector*}
- Disks have **defective** sectors
  - Manufacturers hide this by substituting spare sectors from elsewhere on disk
- Number of sectors per track
  - Not constant in some drives

Rotation speed and density of disks

- Density of bits is uniform
  - Outer tracks have greater length
    - More sectors = greater capacity
  - Drive increases rotation speed as it moves from inner track to outer ones
    - **Constant linear velocity**
- Disk rotation can stay constant
  - Density of bits decreases from inner tracks to outer ones

27

28

Disk scheduling: Objectives

- Fast **access times**
  - *Seek time*
    - Move disk arm to the right cylinder
  - *Rotational latency*
    - Rotate to the desired sector
- Disk **bandwidth**
  - Transfer rates

Disk scheduling: Premise

- With many processes, there are many **pending** disk I/O requests
- Improve access time and bandwidth
  - By managing the **order** in which disk I/O requests are serviced

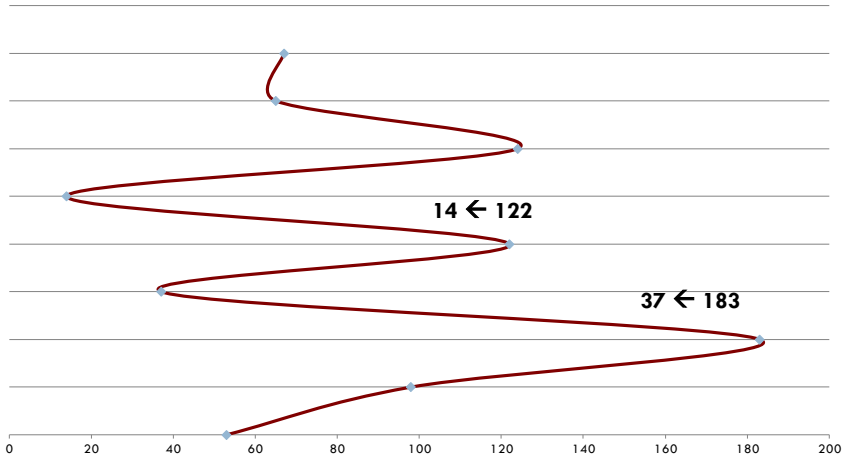
Disk I/O request string

- We consider requests to I/O blocks on **cylinders (or tracks)**
- {98, 183, 37, 122, 14, 124, 65, 67}
  - Initial disk head position: 53
  - Number of cylinders: 200
    - 0-199

First Come Fist Served Scheduling

- Process these requests in the order they arrive
- {98, 183, 37, 122, 14, 124, 65, 67}
- Wild swings
  - 183 → 37, 122 → 14
- Total disk-head movement is way too high
  - In our example: 640

Plotting the disk head movement: FCFS



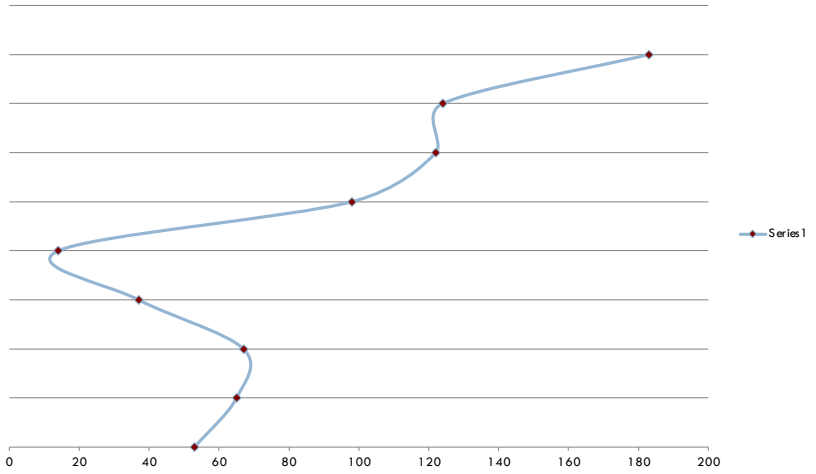
33

Shortest-Seek-Time-First (SSTF) targets disk head movement

- Select request with the **lowest seek time**
  - Service requests closest to current disk head position
- {98, 183, 37, 122, 14, 124, 65, 67}
  - Initially at 53
- {65, 67, 37, 14, 98, 122, 124, 183}
  - Total disk-head movement= 236 cylinders

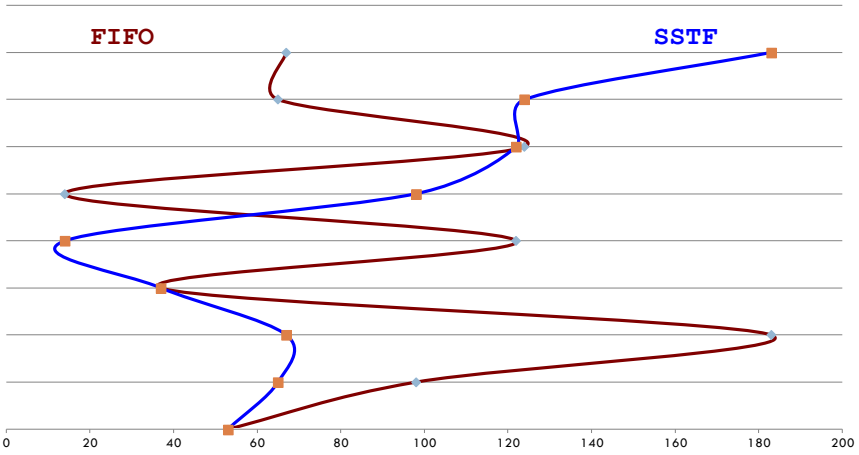
34

Plotting the disk head movement: SSTF



35

Contrasting disk head movements in FCFS and SSTF



36

SSTF

- This is a form of shortest-job-first
- Can cause **starvation** in some requests
- Not optimal
  - SSTF: {65, 67, 37, 14, 98, 122, 124, 183}
    - Total head movement: 236 cylinders
  - We can do better
    - Could have done: {53, 37, 14, 65, 67, 98, 122, 124, 183}
    - Total head movement: 208 cylinders

37

SCAN SCHEDULING AND VARIANTS

38

SCAN scheduling

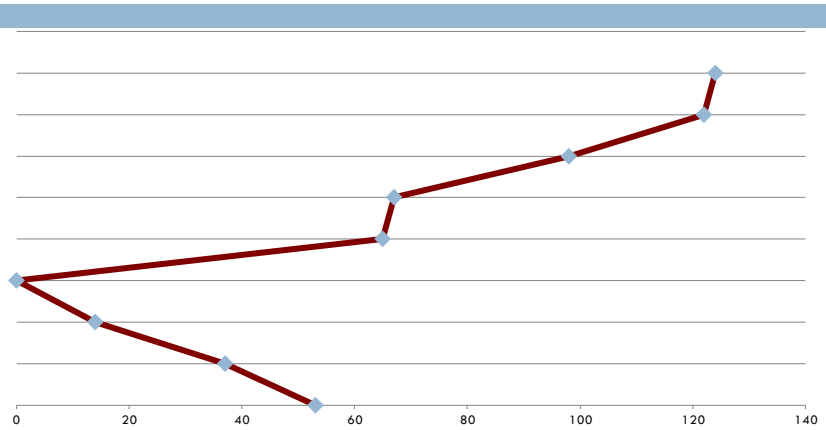
- Start at one end of disk & move to the other end
  - Servicing requests
- Reverse directions at the other end
- Also called **elevator** algorithm

Before applying the algorithm to our schedule

- We need to know
  - The disk head's current **position**
  - **Direction** of the head movement
- {98, 183, 37, 122, 14, 124, 65, 67}
  - Initially at 53
  - Disk arm is moving towards 0

- {14, 37, 65, 67, 98, 122, 124, 183}

Plot of the disk head movement in SCAN scheduling

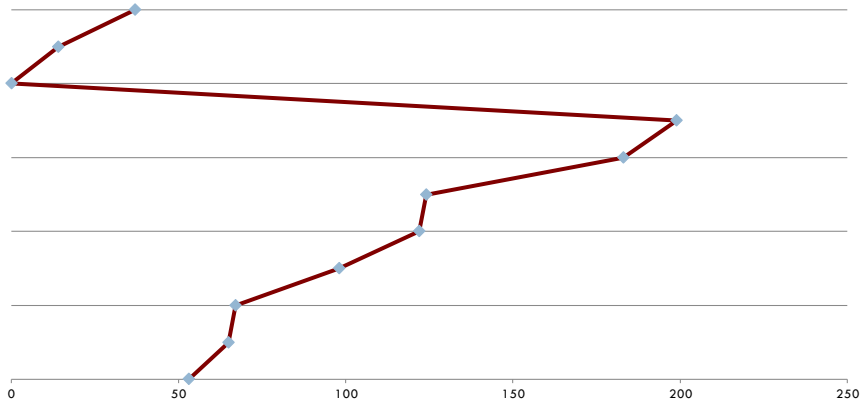


SCAN scheduling

- When requests have been serviced
  - From one end to other
- When the disk head reaches one end
  - Heaviest **density** of requests is at the other end
  - The requests have also waited the **longest**
    - Go there first?
    - Circular SCAN (**C-SCAN**)

Plot of the disk head movement in C-SCAN scheduling

{ 53, 65, 67, 98, 122, 124, 183, **199**, **0**, 14, 37 }

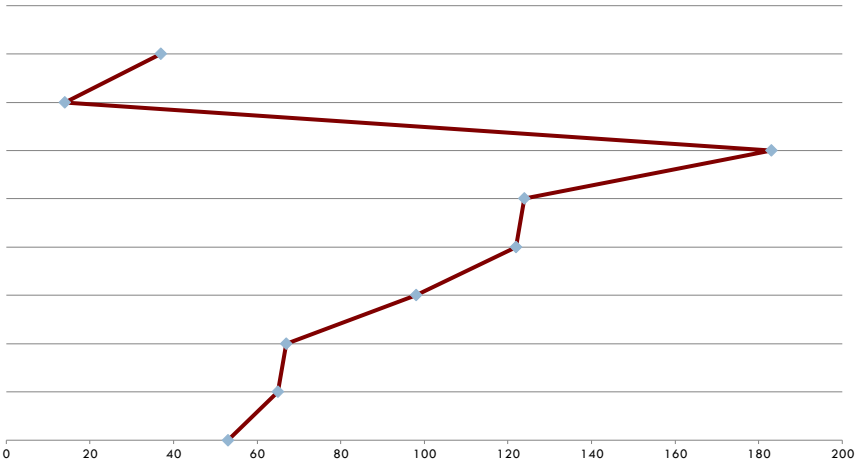


LOOK scheduling is a variant of C-SCAN scheduling

- Arm goes only as far as the final request
- Reverse direction
  - Without going all the way to the end

Plot of the disk head movement in LOOK scheduling

{ 53, 65, 67, 98, 122, 124, 183, ~~199~~, ~~0~~, 14, 37 }



Selection of disk scheduling algorithms

- Depends on **number** and **types** of requests
  - If there is always *just one outstanding request*?
    - All algorithms behave the same way
- Requests for disk service influenced by **file-allocation** method
  - Location of directories and index blocks

OS has other constraints on the order in which these requests are serviced

- **Demand paging** has priority over application I/O
- If **cache is running out of free pages**
  - Writes are more important than reads
- Order a set of disk writes
  - To make file system more robust to system crashes

47

FINAL EXAM

48

Final Exam

Processes and IPC	5
Threads:	10
CPU Scheduling:	10
Process Synchronization & Atomic Transactions	15
Deadlocks	10
Memory Management	10
Virtual Memory	10
Virtualization	10
File Systems	15
Mass Storage & Disk Scheduling	5

The contents of this slide-set are based on the following references

- *Andrew S Tanenbaum. Modern Operating Systems. 4th Edition, 2014. Prentice Hall. ISBN: 013359162X/ 978-0133591620*
- *Avi Silberschatz, Peter Galvin, Greg Gagne. Operating Systems Concepts, 9th edition. John Wiley & Sons, Inc. ISBN-13: 978-1118063330.* [Chapter 1]
- *Kay Robbins & Steve Robbins. Unix Systems Programming, 2nd edition, Prentice Hall ISBN-13: 978-0-13-042411-2.* [Chapter 1]