TCSS 422: OPERATING SYSTEMS

Beyond Physical Memory, I/O Devices, Hard Disk Drives

Wes J. Lloyd
School of Engineering and Technology
University of Washington - Tacoma

June 1, 2023 TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

1

COURSE EVALUATION: TCSS 422 A SPRING 2023

- Please complete the course evaluation survey at: https://uwt.iasystem.org/survey/112275
- **Special features this quarter in TCSS 422:**
- Class sessions LIVE streamed over Zoom, with all lecture recordings made available shortly after class
- No mandatory graded in class activities this quarter to maximize attendance/participation flexibility *(enables mostly asynchronous participation)*
- OBS Studio software used to provide different “scenes” that integrate screen captures with camera, chat, and displays
- Slide refinements to improve online delivery
- Assignment 3 graded as a Quiz/Tutorial: Kernel Module programming
- Extra credit for paperless daily feedback surveys
- Tutorial 3 on File Systems optional for extra credit
- Assignment 2–single producer, single consumer, multiple buffer provides revised scope having less C coding, with primary focus on pthreads, locking, and bounded buffer

June 1, 2023 | TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma | L19.2

2

OBJECTIVES – 6/1

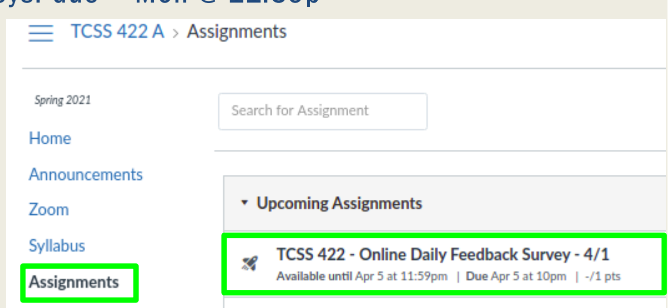
- **Questions from 5/30**
- Assignment 2 – June 2
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Memory Segmentation Activity + answers (available in Canvas)
- Quiz 4 – Page Tables - Due June 8 @ 11:59am
- Final exam – June 8 @ 3:40pm
- Tutorial 3 - File Systems (Optional, Extra Credit)
- Ch. 36 I/O Devices, Ch. 37 Hard Disk Drives
- Practice Final Exam

June 1, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L19.3

3

ONLINE DAILY FEEDBACK SURVEY

- Daily Feedback Quiz in Canvas – Available After Each Class
- Extra credit available for completing surveys ***ON TIME***
- Tuesday surveys: due by ~ Wed @ 11:59p
- Thursday surveys: due ~ Mon @ 11:59p



June 1, 2023	TCSS422: Computer Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L19.4

4

TCSS 422 - Online Daily Feedback Survey - 4/1

Quiz Instructions

Question 1 — 0.5 pts

On a scale of 1 to 10, please classify your perspective on material covered in today's class:

1	2	3	4	5	6	7	8	9	10
Mostly Review To Me				Equal New and Review					Mostly New to Me

Question 2 — 0.5 pts

Please rate the pace of today's class:

1	2	3	4	5	6	7	8	9	10
Slow				Just Right					Fast

5

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (40 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.71 (↓ - previous 6.80)**

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.88 (↑ - previous 5.70)**

6

FEEDBACK FROM 5/30

- Average Memory Access Time (AMAT) remains least clear to me…
- The average memory access time (AMAT) metric is used to show the impact of memory page hit-to-miss ratios
- The goal is to retain pages in memory to obtain the highest number of "hits", in order to ***minimize page faults***
- The metric shows how increasing the number of misses greatly increases memory access time, especially if the swap space is support on a legacy hard disk drive (HDD) with 80x access speed.
  - Today's SSDs will be closer to 2 to 4x DRAM access speed
  - Page faults have less impact on AMAT, but it is still important

7

AVERAGE MEMORY ACCESS TIME (AMAT)

- Average memory access time (AMAT) can be estimated:

$$AMAT = (P_{Hit} * T_M) + (P_{Miss} * T_D)$$

Argument	Meaning
T_M	The cost of accessing memory (time)
T_D	The cost of accessing disk (time)
P_{Hit}	The probability of finding the data item in the cache(a hit)
P_{Miss}	The probability of not finding the data in the cache(a miss)

- Consider T_M = 100 ns, T_D = 10ms
- Consider P_{hit} = .9 (90%), P_{miss} = .1
- Consider P_{hit} = .999 (99.9%), P_{miss} = .001

8

OBJECTIVES – 6/1

- Questions from 5/30
- **Assignment 2 – June 2**
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Memory Segmentation Activity + answers (available in Canvas)
- Quiz 4 – Page Tables - Due June 8 @ 11:59am
- Final exam – June 8 @ 3:40pm
- Tutorial 3 - File Systems (Optional, Extra Credit)
- Ch. 36 I/O Devices, Ch. 37 Hard Disk Drives
- Practice Final Exam

June 1, 2023 | TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma | L19.9

9

OBJECTIVES – 6/1

- Questions from 5/30
- Assignment 2 – June 2
- **Assignment 3: (Tutorial) Introduction to Linux Kernel Modules**
- Memory Segmentation Activity + answers (available in Canvas)
- Quiz 4 – Page Tables - Due June 8 @ 11:59am
- Final exam – June 8 @ 3:40pm
- Tutorial 3 - File Systems (Optional, Extra Credit)
- Ch. 36 I/O Devices, Ch. 37 Hard Disk Drives
- Practice Final Exam

June 1, 2023 | TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma | L19.10

10

ASSIGNMENT 3: INTRODUCTION TO LINUX KERNEL MODULES

- Assignment 3 provides an introduction to kernel programming by demonstrating how to create a Linux Kernel Module
- Kernel modules are commonly used to write device drivers and can access protected operating system data structures
  - For example: Linux `task_struct` process data structure
- Assignment 3 is scored in the Quizzes / Activities / Tutorials category
  - Lowest two grades in this category are dropped

11

OBJECTIVES – 6/1

- Questions from 5/30
- Assignment 2 – June 2
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- **Memory Segmentation Activity + answers (available in Canvas)**
- Quiz 4 – Page Tables - Due June 8 @ 11:59am
- Final exam – June 8 @ 3:40pm
- Tutorial 3 - File Systems (Optional, Extra Credit)
- Ch. 36 I/O Devices, Ch. 37 Hard Disk Drives
- Practice Final Exam

12

OBJECTIVES – 6/1

- Questions from 5/30
- Assignment 2 – June 2
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Memory Segmentation Activity + answers (available in Canvas)
- **Quiz 4 – Page Tables - Due June 8 @ 11:59am**
- Final exam – June 8 @ 3:40pm
- Tutorial 3 - File Systems (Optional, Extra Credit)
- Ch. 36 I/O Devices, Ch. 37 Hard Disk Drives
- Practice Final Exam

June 1, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L19.13

13

OBJECTIVES – 6/1

- Questions from 5/30
- Assignment 2 – June 2
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Memory Segmentation Activity + answers (available in Canvas)
- Quiz 4 – Page Tables - Due June 8 @ 11:59am
- **Final exam – June 8 @ 3:40pm**
- Tutorial 3 - File Systems (Optional, Extra Credit)
- Ch. 36 I/O Devices, Ch. 37 Hard Disk Drives
- Practice Final Exam

June 1, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L19.14

14

FINAL EXAM – THURSDAY JUNE 8 @ 3:40PMTH

- Thursday June 8 from 3:40 to 5:40 pm
  - Final (100 points)
  - SHORT: similar number of questions as the midterm
  - 2-hours
  - Focus on new content - since the midterm (~70% new, 30% before)
- Final Exam Review -
  - Complete Memory Segmentation Activity
  - Complete Quiz 4
  - Practice Final Exam Questions – 2nd hour of June 1st class session
  - Individual work
  - 2 pages of notes (any sized paper), double sided
  - Basic calculators allowed
  - NO smartphones, laptop, book, Internet, group work

OBJECTIVES – 6/1

- Questions from 5/30
- Assignment 2 – June 2
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Memory Segmentation Activity + answers (available in Canvas)
- Quiz 4 – Page Tables - Due June 8 @ 11:59am
- Final exam – June 8 @ 3:40pm
- **Tutorial 3 - File Systems (Optional, Extra Credit)**
- Ch. 36 I/O Devices, Ch. 37 Hard Disk Drives
- Practice Final Exam

OBJECTIVES – 6/1

- Questions from 5/30
- Assignment 2 – June 2
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Memory Segmentation Activity + answers (available in Canvas)
- Quiz 4 – Page Tables - Due June 8 @ 11:59am
- Final exam – June 8 @ 3:40pm
- Tutorial 3 - File Systems (Optional, Extra Credit)
- **Ch. 36 I/O Devices**, Ch. 37 Hard Disk Drives
- Practice Final Exam

17

CHAPTER 36: I/O DEVICES

18

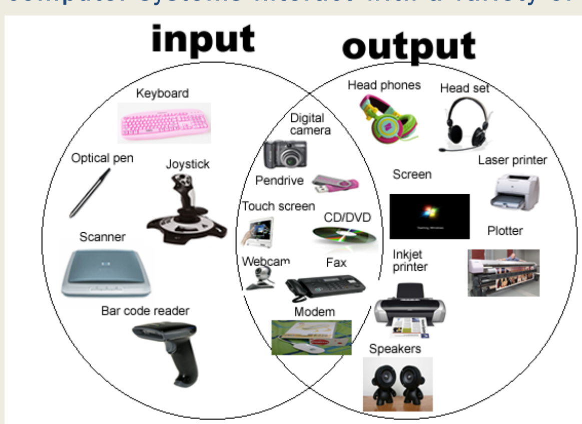
OBJECTIVES

- Chapter 36
  - I/O: Polling vs Interrupts
  - Programmed I/O (PIO)
    - Port-mapped I/O (PMIO)
    - Memory-mapped I/O (MMIO)
  - Direct memory Access (DMA)

19

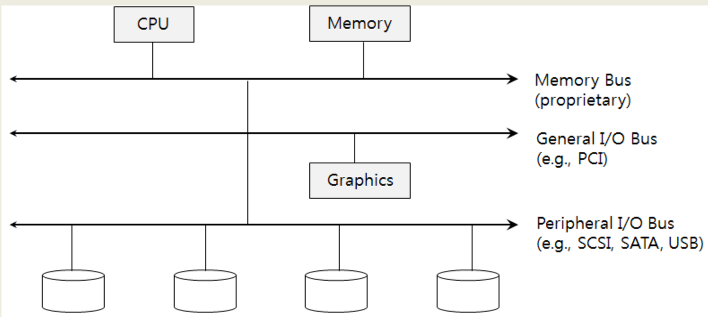
I/O DEVICES

- Modern computer systems interact with a variety of devices



20

COMPUTER SYSTEM ARCHITECTURE



Prototypical System Architecture

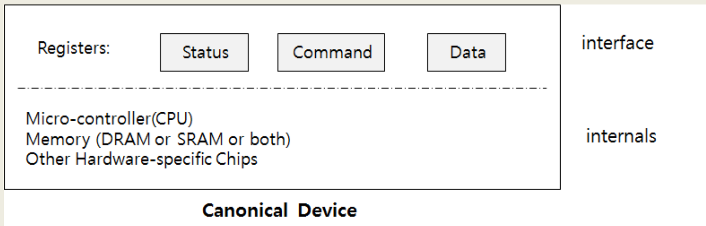
VERY FAST: CPU is attached to main memory via a Memory bus.
FAST: High speed devices (e.g. video) are connected via a General I/O bus.
SLOWER: Disks are connected via a Peripheral I/O bus.

I/O BUSES

- Buses
  - Buses closer to the CPU are faster
  - Can support fewer devices
  - Further buses are slower, but support more devices
- Physics and costs dictate "levels"
  - Memory bus
  - General I/O bus
  - Peripheral I/O bus
- Tradeoff space: speed vs. locality

CANONICAL DEVICE

- Consider an arbitrary canonical ***"standard/generic"*** device



- Two primary components
  - Interface (registers for communication)
  - Internals: Local CPU, memory, specific chips, firmware (embedded software)

CANONICAL DEVICE: HARDWARE INTERFACE

- Status register
  - Maintains current device status
- Command register
  - Where commands for interaction are sent
- Data register
  - Used to send and receive data to the device

General concept:
The OS interacts and controls device behavior by reading and writing the device registers.

OS DEVICE INTERACTION

- Common example of device interaction

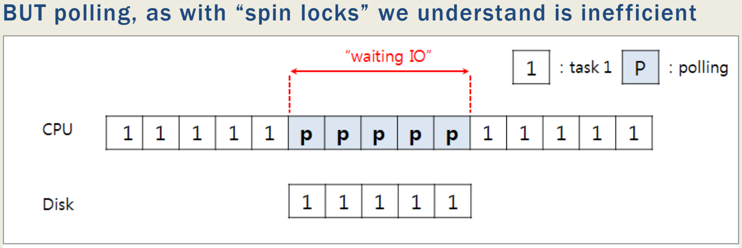
```
while ( STATUS == BUSY)
    ; //wait until device is not busy
write data to data register
write command to command register
    Doing so starts the device and executes the command
while ( STATUS == BUSY)
    ; //wait until device is done with your request
```

Poll- Is device available?
Command parameterization
Send command
Poll – Is device done?

25

POLLING

- OS checks if device is *READY* by repeatedly checking the STATUS register
  - Simple approach
  - CPU cycles are wasted without doing meaningful work
  - Ok if only a few cycles, for rapid devices that are often READY
  - BUT polling, as with "spin locks" we understand is inefficient



CPU utilization by polling

26

INTERRUPTS VS POLLING

- For longer waits, put process waiting on I/O to sleep
- Context switch (C/S) to another process
- When I/O completes, fire an interrupt to initiate C/S back
  - Advantage: better multi-tasking and CPU utilization
  - Avoids: unproductive CPU cycles (polling)

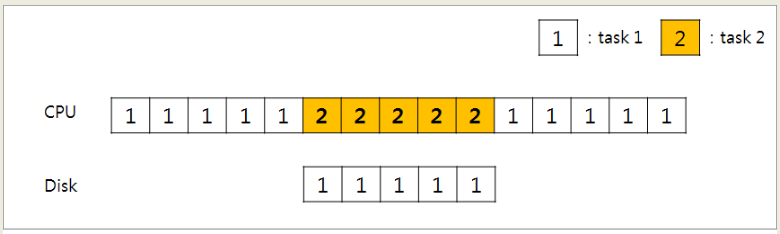


Diagram of CPU utilization by interrupt

27

INTERRUPTS VS POLLING - 2

What is the tradeoff space ?

- Interrupts are not always the best solution
  - How long does the device I/O require?
  - What is the cost of context switching?

If device I/O is fast → polling is better.
When I/O time < 1 CPU time slice (e.g. 10 ms)

If device I/O is slow → interrupts are better.
When I/O time > 1 CPU time slice

28

INTERRUPTS VS POLLING - 3

- Alternative: two-phase hybrid approach
  - Initially poll, then sleep and use interrupts
- Issue: livelock problem
  - Common with network I/O
  - Many arriving packets generate ***many many*** interrupts
  - Overloads the CPU!
  - No time to execute code, just interrupt handlers !
- Livelock optimization
  - Coalesce multiple arriving packets (for different processes) into fewer interrupts
  - Must consider number of interrupts a device could generate

DEVICE I/O

- To interact with a device we must send/receive DATA
- Two general approaches:
  - Programmed I/O (PIO):
    - Port mapped I/O (PMIO)
    - Memory mapped I/O (MMIO)
  - Direct memory access (DMA)

Transfer Modes

Mode	#	Maximum transfer rate (MB/s)	cycle time
PIO	0	3.3	600 ns
	1	5.2	383 ns
	2	8.3	240 ns
	3	11.1	180 ns
	4	16.7	120 ns
Single-word DMA	0	2.1	960 ns
	1	4.2	480 ns
	2	8.3	240 ns
Multi-word DMA	0	4.2	480 ns
	1	13.3	150 ns
	2	16.7	120 ns
	3[34]	20	100 ns
	4[34]	25	80 ns
Ultra DMA	0	16.7	240 ns ÷ 2
	1	25.0	160 ns ÷ 2
	2 (Ultra ATA/33)	33.3	120 ns ÷ 2
	3	44.4	90 ns ÷ 2
	4 (Ultra ATA/66)	66.7	60 ns ÷ 2
	5 (Ultra ATA/100)	100	40 ns ÷ 2
	6 (Ultra ATA/133)	133	30 ns ÷ 2
	7 (Ultra ATA/167)[35]	167	24 ns ÷ 2

From https://en.wikipedia.org/wiki/Parallel_ATA

31

PROGRAMMED I/O (PIO)

- I/O performed on the CPU
- CPU time is consumed performing I/O
- CPU supports data movement (input/output)
- PIO is slow: CPU is occupied with meaningless work

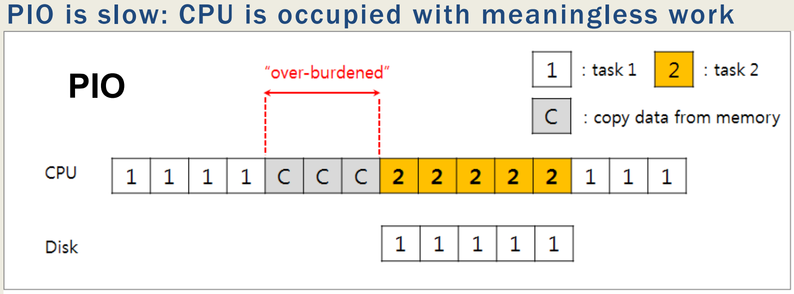


Diagram of CPU utilization

32

PIO DEVICES

- Legacy serial ports
- Legacy parallel ports
- PS/2 keyboard and mouse
- Legacy MIDI, joysticks
- Old network interfaces

PROGRAMMED I/O DEVICE (PIO) INTERACTION

- Two primary PIO methods
  - Port mapped I/O (PMIO)
  - Memory mapped I/O (MMIO)

PORT MAPPED I/O (PMIO)

- Device specific CPU I/O Instructions
- Follows a Complex Instruction Set - CISC model (Intel):
- Specific CPU instructions are used for device I/O
- x86/x86-64: `in` and `out` instructions
- `outb`, `outw`, `outl`
- 1, 2, 4 byte copy from EAX → device's I/O port

MEMORY MAPPED I/O (MMIO)

- Device's memory is mapped to standard memory addresses
- MMIO is common with RISC CPUs:
  Special CPU instructions for PIO eliminated
- Old days: 16-bit CPUs didn't have a lot of spare memory space
- Today's CPUs have LARGE address spaces:
  32-bit (4GB addr space) & 64-bit (256 TB addr space)
- Device I/O uses regular CPU instructions usually used to read/write memory to access device
- Device is mapped to unique memory address **reserved** for I/O
  - Address must not be available for normal memory operations.
  - Generally very high addresses (out of range of type addresses)
- Device monitors CPU address bus and respond to instructions on their addresses

DIRECT MEMORY ACCESS (DMA)

- Copy data in memory by ***offloading*** to "DMA controller"
- Many devices (including CPUs) integrate DMA controllers
- CPU gives DMA: memory address, size, and copy instruction
- DMA performs I/O independent of the CPU
- DMA controller generates CPU interrupt when I/O completes

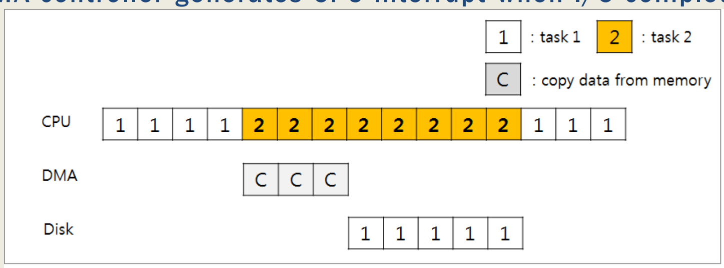


Diagram of CPU utilization by DMA

DIRECTORY MEMORY ACCESS – 2

- Many devices use DMA
  - HDD/SSD controllers (ISA/PCI)
  - Graphics cards
  - Network cards
  - Sound cards
  - Intra-chip memory transfer for multi-core processors

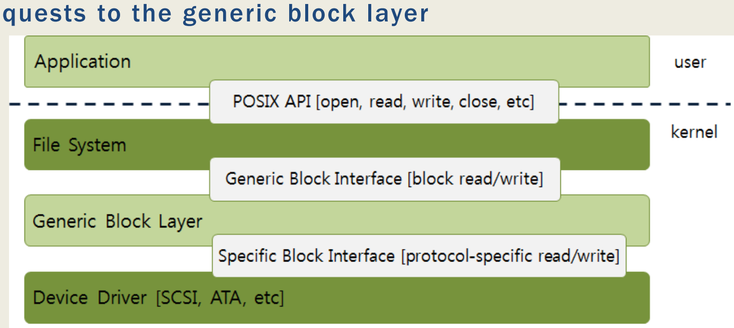
- DMA allows computation and data transfer time to proceed in parallel

DEVICE INTERACTION

- The OS must interact with a variety of devices
- Example: Consider a file system that works across a variety of types of disks:
  - SCSI, IDE, USB flash drive, DVD, etc.
- File system should be general purpose, where device specific I/O implementation details are abstracted
- **Device drivers** use abstraction to provide general interfaces for vendor specific hardware
- In Linux: block devices

FILE SYSTEM ABSTRACTION

- Layered approach to I/O abstraction in Linux
- C functions (open, read, write) issue **block read** and **write** requests to the generic block layer



The File System Stack

I/O DEVICE ABSTRACTION ISSUES

- **Too much abstraction**
  - Many devices provide special capabilities
  - Example: SCSI Error handling
  - SCSI devices provide extra details which are lost to the OS when using generic device drivers
  - Printers may use abstract (generic) device drivers resulting in inaccessibility of custom features
- **Buggy device drivers**
  - 70% of OS code is in device drivers
  - Device drivers are required for every device plugged in
  - Drivers are often 3rd party, which is not quality controlled at the same level as the OS (Linux, Windows, MacOS, etc.)

June 1, 2023 | TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma | L19.41

41

WE WILL RETURN AT 4:50PM

June 1, 2023 | TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma | L19.42

42

OBJECTIVES – 6/1

- Questions from 5/30
- Assignment 2 – June 2
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Memory Segmentation Activity + answers (available in Canvas)
- Quiz 4 – Page Tables - Due June 8 @ 11:59am
- Final exam – June 8 @ 3:40pm
- Tutorial 3 - File Systems (Optional, Extra Credit)
- Ch. 36 I/O Devices, Ch. 37 Hard Disk Drives
- Practice Final Exam

43

CH. 37: HARD DISK DRIVES

44

OBJECTIVES

- Chapter 37
  - HDD Internals
  - Seek time
  - Rotational latency
  - Transfer speed
  - Capacity
  - Scheduling algorithms

June 1, 2023 | TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma | L19.45

45

HARD DISK DRIVE (HDD)

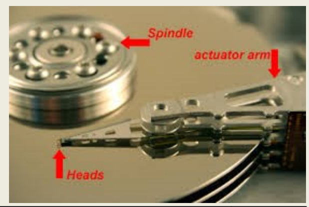
- Primary means of data storage (persistence) for decades
  - Remains inexpensive for high capacity storage
  - 2020: 16 TB HDD - $400, ~15.3 TB SSD - $4,380
- Consists of a large number of data **sectors**
- Sector size is 512-bytes
- An n sector HDD
  can be is addressed as an array of 0..n-1 sectors

June 1, 2023 | TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma | L19.46

46

HDD INTERFACE

- Writing disk sectors is atomic (512 bytes)
- Sector writes are completely successful, or fail
- Many file systems will read/write 4KB at a time
  - Linux ext3/4 default filesystem blocksize – 4096
- Same as typical memory page size



June 1, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L19.47

47

BLOCK SIZE IN LINUX EXT4

- `mkefs.ext4  -i <bytes-per-inode>`
- Formats disk w/ ext4 filesys with specified byte-to-inode ratio
- Today's disks are so large, some use cases with many small files can run out of inodes before running out of disk space
- Each inode record tracks a file on the disk
- Larger bytes-per-inode ratio results in fewer inodes
  - Default is around ~4096
- Value shouldn't be smaller than blocksize of filesystem
- <u>**Note:**</u> It is not possible to expand the number of inodes after the filesystem is created, - be careful deciding the value
- Check inode stats: `tune2fs -l /dev/sda1` *(← disk dev name)*

June 1, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L19.48

48

EXAMPLE: USDA SOIL EROSION MODEL WEB SERVICE (RUSLE2)

- Host ~2,000,000 small XML files totaling 9.5 GB on a ~20GB filesystem on a cloud-based Virtual Machine
- With default inode ratio (4096 block size), only ~488,000 files will fit
- Drive less than half full, but files will not fit !
- HDDs support a minimum block size of 512 bytes
- OS filesystems such as ext3/ext4 can support "finer grained" management at the expense of a larger catalog size
  - Small inode ratio- inodes will considerable % of disk space

EXAMPLE: USDA SOIL EROSION MODEL WEB SERVICE (RUSLE2) - 2

- Free space in bytes (df)

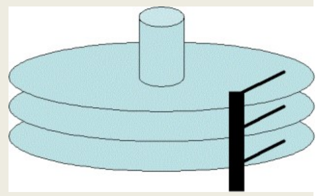
Device	total size	bytes-used	bytes-free	usage	
/dev/vda2	13315844	9556412	3049188	76%	/mnt

- Free inodes (df –i) @ 512 bytes / node

Device	total inodes	used	free	usage	
/dev/vda2	3552528	1999823	1552705	57%	/mnt

HDD INTERFACE - 2

- Torn write
  - When OS uses larger block size than HDD
  - Block writes not **atomic** - they SPAN multiple HDD sectors
  - Upon power failure only a portion of the OS block is written – *can lead to data corruption…*
- HDD access
  - Sequential reads of sectors is fastest
  - Random sector reads are slow
  - Disk head continuously must jump to different tracks

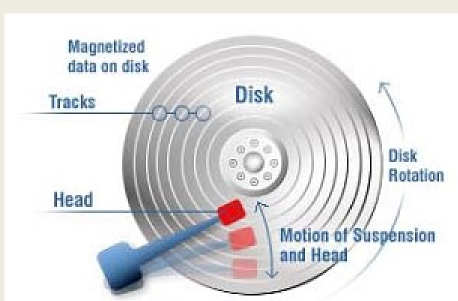


June 1, 2023 | TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma | L19.51

51

HDD PLATTER

- Made from aluminum coated with thin magnetic layer
- HDD records on both sides of each platter
- Data is stored by inducing magnetic changes

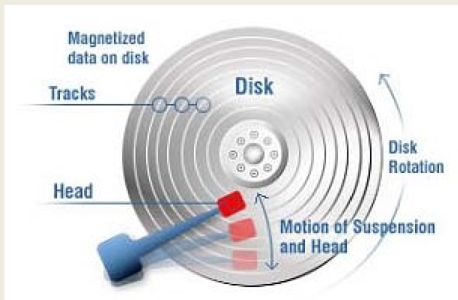


June 1, 2023 | TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma | L19.52

52

HDD SPINDLE

- Connected to motor which spins the disk
- Speed measures in RPM (rotations per minute)
- Typical: 7200-15000 rpm
- 10000 rpm – 1 rotation in 6ms; 15k rpm 1 rotation in 4ms



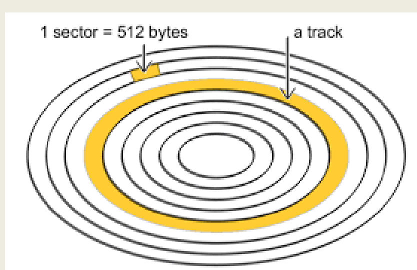
June 1, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L19.53

53

HDD TRACK

- Concentric circle of sectors
- Single side of platter contains 290 K tracks (2008)
- Zones: groups of tracks with same # of sectors

Outer tracks have More sectors

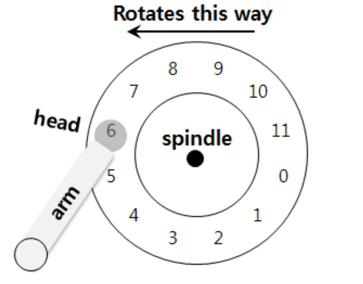


June 1, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L19.54

54

EXAMPLE: SIMPLE DISK DRIVE

- Single track disk
- Head: one per surface of drive
- Arm: moves heads across surface of platters

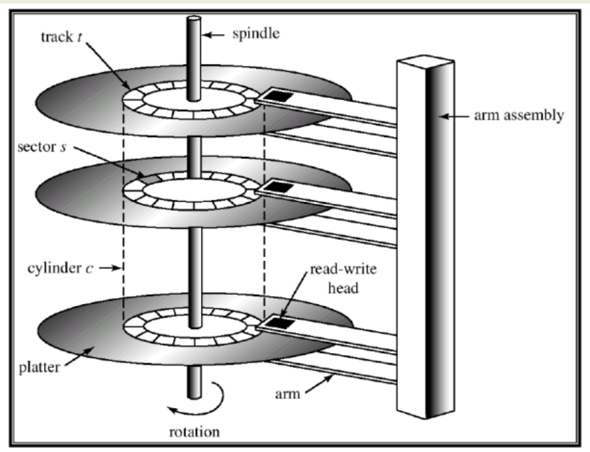


A Single Track Plus A Head

June 1, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L19.55

55

HARD DISK STRUCTURE



June 1, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L19.56

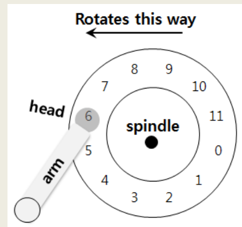
56

SINGLE-TRACK LATENCY: THE ROTATIONAL DELAY

- **Rotational latency** ($T_{rotation}$): time to rotate to desired sector
- Average $T_{rotation}$ is ~ about half the time of a full rotation
- How to calculate $T_{rotation}$ from rpm

1. Calculate time for 1 rotation based on rpm
   > Convert rpm to rps
2. Divide by two (*average rotational latency*)

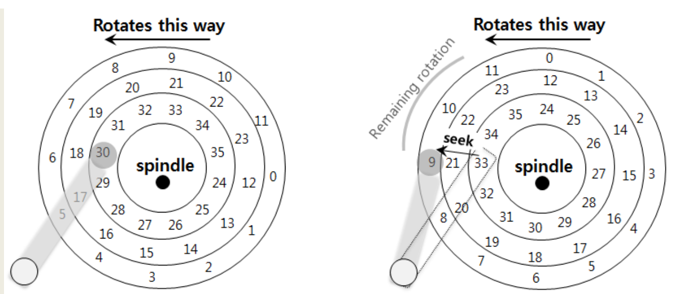
- 7200rpm = 8.33ms per rotation /2= ~4.166ms
- 10000rpm = 6ms per rotation /2= ~3ms
- 15000rpm = 4ms per rotation /2= ~2ms



A Single Track Plus A Head

57

SEEK TIME



Three Tracks Plus A Head (Right: With Seek)
(e.g., read to sector 11)

- **Seek time** (T_{seek}): time to move disk arm to proper track
- Most time consuming HDD operation

58

FOUR PHASES OF SEEK

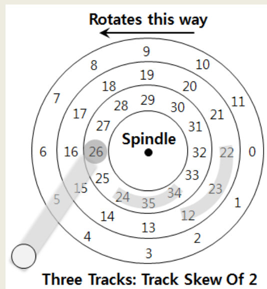
- Acceleration → coasting → deceleration →settling
- **Acceleration**: the arm gets moving
- **Coasting**: arm moving at full speed
- **Deceleration**: arm slow down
- **Settling**: Head is carefully positioned over track
  - Settling time is often high, from .5 to 2ms

HDD I/O

- Data transfer
  - Final phase of I/O: time to read or write to disk surface
- Complete I/O cycle:
  1. Seek (accelerate, coast, decelerate, settle)
  2. Wait on rotational latency (*until track aligns*)
  3. Data transfer

TRACK SKEW

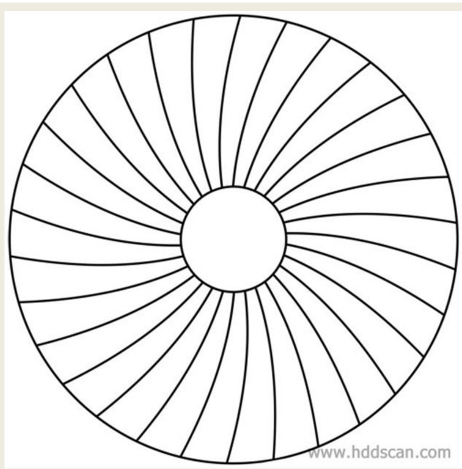
- Sectors are offset across tracks to allow time for head to reposition for sequential reads
- Without track skew, when head is repositioned sector would have already been passed



June 1, 2023 | TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma | L19.61

61

TRACK SKEW - 2



June 1, 2023 | TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma | L19.62

62

HDD CACHE

- Buffer to support caching reads and writes
- Improves drive response time
- Up to 256 MB, slowly have been growing
- Two styles
  - Writeback cache
    - Report write complete immediately when data is transferred to HDD cache
    - Dangerous if power is lost
  - Writethrough cache
    - Reports write complete only when write is physically completed on disk

TRANSFER SPEED

- Can calculate I/O transfer speed with:
- I/O Time: $T_{I/O} = T_{seek} + T_{rotation} + T_{transfer}$
- $T_{transfer} = DATA_{size} \times Rate_{I/O}$
- Rate of I/O: $R_{I/O} = \frac{Size_{Transfer}}{T_{I/O}}$

EXAMPLE: I/O SPEED

- Compare two disks:
1. Random workload: 4KB (random read on HDD)
2. Sequential workload: 100MB (contiguous sectors)
   > Calculate $T_{rotation}$ from rpm (rpm→rps, time for 1 rotation / 2)

	Cheetah 15K.5	Barracuda
Capacity	300 GB	1 TB
RPM	15,000	7,200
Average Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s
Platters	4	4
Cache	16 MB	16/32 MB
Connects Via	SCSI	SATA

Disk Drive Specs: SCSI Versus SATA

65

EXAMPLE: I/O SPEED

1. Random workload: 4KB (random read on HDD)
2. Sequential workload: 100MB (contiguous sectors)

$$T_{I/O} = T_{seek} + T_{rotation} + T_{transfer}$$

$$T_{transfer} = Data_{size} \ x \ Rate_{I/O}$$

$$R_{I/O} = \frac{Size_{Transfer}}{T_{I/O}}$$

		Cheetah 15K.5	Barracuda
	T_{seek}	4 ms	9 ms
	$T_{rotation}$	2 ms	4.2 ms
4 KB Random	$T_{transfer}$	30 microsecs	38 microsecs
	$T_{I/O}$	6 ms	13.2 ms
	$R_{I/O}$	0.66 MB/s	0.31 MB/s
100 MB Sequential	$T_{transfer}$	800 ms	950 ms
	$T_{I/O}$	806 ms	963.2 ms
	$R_{I/O}$	125 MB/s	105 MB/s

Disk Drive Performance: SCSI Versus SATA

There is a huge gap in drive throughput between random and sequential workloads

66

MODERN HDD SPECS

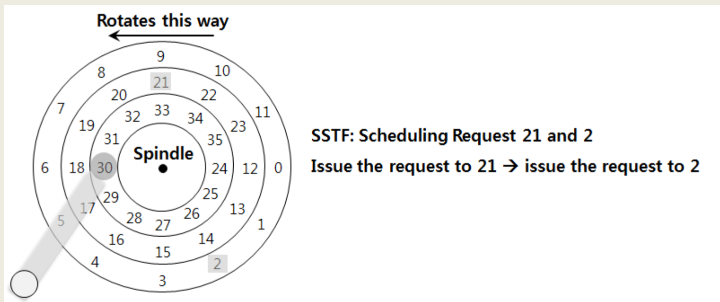
- See sample HDD configurations here:
  - Up to 20 TB
- https://www.westerndigital.com/products/data-center-drives#hard-disk-hdd

DISK SCHEDULING

- Disk scheduler: determine how to order I/O requests
- Multiple levels - OS and HW
- OS: provides ordering
- HW: further optimizes using intricate details of physical HDD implementation and state

SSTF – SHORTEST SEEK TIME FIRST

- Disk scheduling – which I/O request to schedule next
- Shortest Seek Time First (SSTF)
- Order queue of I/O requests by nearest track



69

SSTF ISSUES

- Problem 1: HDD abstraction
- Drive geometry not available to OS. Nearest-block-first is a comparable alternate algorithm.
- Problem 2: Starvation
- Steady stream of requests for local tracks may prevent arm from traversing to other side of platter

70

DISK SCHEDULING ALGORITHMS

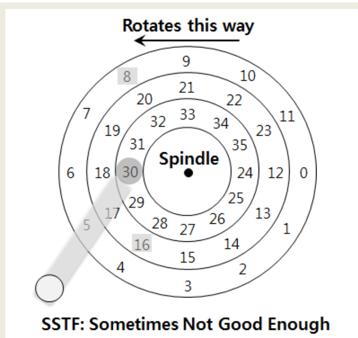
- **SCAN (SWEEP)**
- Perform single repeated passes back and forth across disk
- Issue: if request arrives for a recently visited track it will not be revisited until a full cycle completes

- **F-SCAN**
- Freeze incoming requests by adding to queue during scan
- Cache arriving requests until later
- Delays help avoid starvation by postponing servicing nearby newly arriving requests vs. requests at edge of sweep
- Provides better fairness

- **Elevator (C-SCAN)** – circular scan
- Sweep only one direction (e.g. outer to inner) and repeat
- SCAN favors middle tracks vs. outer tracks with 2-way sweep

SHORTEST TIME POSITIONING FIRST

- Determine next sector to read?
  - Where: $T_{seek} = T_{rotation}$
- On which track?
- On which sector?



On modern drives, both seek and rotation are roughly equivalent:
Thus, SPTF (Shortest Positioning Time First) is useful.

OPTIMIZATION: I/O MERGING

- Group temporary adjacent requests
- Reduce overhead
- Read (memory blocks): 33 8 34
- How long we should wait for I/O ?
- When do we know we have waited too long?

June 1, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L19.73

73

OBJECTIVES – 6/1

- Questions from 5/30
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Memory Segmentation Activity + answers (available in Canvas)
- Quiz 4 – Page Tables - Due June 8 @ 11:59am
- Final exam – June 8 @ 3:40pm
- Tutorial 3 - File Systems (Optional, Extra Credit)
- Ch. 36 I/O Devices, Ch. 37 Hard Disk Drives
- **Practice Final Exam**

June 1, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L19.74

74

PRACTICE FINAL EXAM QUESTIONS

75

QUESTION 1 – BASE AND BOUNDS

- A computer system uses a simple base/bounds register pair to virtualize address spaces. For each traces fill in the missing values of virtual addresses, physical addresses, base, and/or bounds registers. In some cases, it is not possible to provide an exact value. If so, specify a range (e.g. greater than 100), or value that is not a single number.

Scenario 1

Virtual Address	Physical Address		
100	600		
300	800	Base?	____________
699	1199		
700	[fault]	Bounds?	____________

76

Q1 - 2

Scenario 2

Virtual Address	Physical Address		
300	1500	Base?	____________
1600	2800		
1801	___________?	Bounds?	____________
2801	4001		

Scenario 3

Virtual Address	Physical Address		
____________	1000	Base?	1000_________
____________	1100		
____________	2999	Bounds?	2000_________
____________	[fault]		

77

QUESTION 2 – SINGLE-LEVEL PAGE TABLE

- Consider a computer with 4 GB (2^{32}) of physical memory, where the page size is 4 KB (2^{12}). For simplicity assume than 1GB=1000MB, 1MB=1000KB, 1KB=1000 bytes
- (a) How many pages must be tracked by a single-level page table if the computer has 4GB (2^{32}) of physical memory and the page table size is 4 KB (2^{12})?
- (b) How many bits are required for the virtual page number (VPN) to address any page within this 4GB (2^{32}) memory space?
- (c) Assuming that the smallest addressable unit of memory within a page is a byte (8-bits), how many bits are required for the offset to refer to any byte in the 4 KB page?
- (d) Assuming each page table entry (PTE) requires 4 bytes of memory, how much memory is required to store the page table for one process (in MB)?

78

79

Q2 - 2

- (e) Using this memory requirement, how many processes would fill the memory with page table data on a 4GB computer?

June 1, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L19.80

80

QUESTION 3 - TWO-LEVEL PAGE TABLE

- Consider a computer with 1 GB (2^{30}) of physical memory, where the page size is 1024 bytes (1KB) (2^{10}). We would like to index memory pages using a two level page table consisting of a page directory which refers to page tables which are created on demand to index the entire memory space.
- For simplicity assume than 1GB=1000MB, 1MB=1000KB, 1KB=1000 bytes
- (a) For a two-level page table, divide the VPN in half. How many bits are required for the page directory index (PDI) in a two-level scheme?
- (b) How many bits are required for the page table index (PTI)?
- (c) How many bits are required for an offset to address any byte in the 1 KB page?

81

82

Q3 - 2

- (d) Assuming each page table entry (PTE) requires 4 bytes of memory, how many extra bits are available for status bits?

- (e) HelloWorld.c consists of 4 memory pages. One code page, one heap page, one data segment page, and one stack segment page. How large is the two-level page table in bytes with the structure described above that could index the all 4 memory pages of HelloWorld.c?
  Hint: There should be 2 tables, a page directory, and a page table.

- *(f) Assuming the same page table as for HelloWorld.c, using the exact same two-level page table, how large in bytes could the program grow to before needing to expand the page table?*

QUESTION 4 – CACHE TRACING

- Consider a 3-element cache with the cache arrival sequences below.
- Determine the number of cache hits and cache misses using each of the following cache replacement policies:

A. Optimal policy

Arrival sequence:

5 3 7 5 3 1 0 7 1 6 4 3 2 1 3

Working Cache
Cache 1:

Cache 2:

Cache 3:

Hits: ________________

Misses: ________________

85

Q4 - 2

B. FIFO policy

Arrival sequence:

5 3 7 5 3 1 0 7 1 6 4 3 2 1 3

Working Cache
Cache 1:

Cache 2:

Cache 3:

Hits: ________________

Misses: ________________

86

Q4 - 3

C. LRU policy

Arrival sequence:

5 3 7 5 3 1 0 7 1 6 4 3 2 1 3

Working Cache
Cache 1:

Cache 2:

Cache 3:

Hits: ________________

Misses: ________________

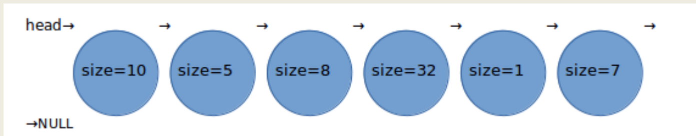
QUESTION 5 – FREE SPACE MANAGEMENT

- Free space management involves capturing a description of the computer's free memory using a data structure, storing this data structure in memory, and OS support to rapidly use this structure to determine an appropriate location for new memory allocations. An efficient implementation is very important when scaling up the number of operations the OS is required to perform.
- Consider the use of a linked list for a free space list where each node is represented by placing the following structure in the header of the memory chunk:

```
typedef struct __node_t
{
    int size;
    struct __node_t *next;
} node_t;
```

Q5 - 2

- Consider the following free space list:



- (a) Consider the **next fit** allocation strategy. For this free list above, how many comparison operations must be performed to identify a free chunk of **30-bytes** ?
- (b) After the last free space identification, the chunk is split and the remaining free space is returned to the free space list. Now, consider the **next fit** allocation strategy. After finding a free space for the previous request, how many comparisons are required to identify a free chunk of 10-bytes?

June 1, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L19.89

89

Q5 - 3

- Now, after the last free space identification the chunk is split and the remaining free space is returned to the free space list. Now consider each of the following free space allocation strategies. How many comparisons are required on the updated free space list to find a free chunk of 2 bytes using:
- (c) best fit?
- (d) worst fit?
- (e) first fit?

June 1, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L19.90

90

QUESTIONS

91