# TCSS 422: OPERATING SYSTEMS

## Paging: Smaller Tables, Beyond Physical Memory

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

May 28, 2020   TCSS422: Operating Systems [Spring 2020]
School of Engineering and Technology, University of Washington - Tacoma

---

## OBJECTIVES – 5/28

- **Questions from 5/26**
- **Tutorial 2 (pthreads, locks, conditions) – due Thurs June 4**
- **Quiz 3 posted – Active Reading Ch. 19 – due Tues June 2**
- **Assignment 2 (based on Ch. 30) – due Sun May 31**
- **Assignment 3 – on Linux kernel programming – offered in "tutorial" format – to be posted ~May 28**
- **Chapter 20: Paging: Smaller Tables**
  - **Smaller Tables, Hybrid Tables, Multi-level Page Tables**
- **Chapter 21/22: Beyond Physical Memory**
  - **Swapping Mechanisms**
  - **Swapping Policies**
- **Chapter 36/37: I/O Devices, Hard Disk Drives**

| May 28, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.2 |

---

## MATERIAL / PACE

- Please classify your perspective on material covered in today's class (38 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.62 (↑ from 6.29)**

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.84 (↑ from 5.70)**

| May 28, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.3 |

---

**Have you upgraded to the new Zoom 5.0 client?**
**(required starting May 30)**

Yes

Am planning to upgrade soon

What? There's an upgrade?

I tried, but had problems upgrading

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

---

## FEEDBACK FROM 5/26

| May 28, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.5 |

---

## ASSIGNMENT 2 – EXTRA CREDIT

- Please remember to add comments to pcMatrix.c to indicate if extra credit should be graded for Assignment #2:

> **\* - EXTRA CREDIT- COMMENTS ARE REQUIRED:**
> Comments must be included at the top of the mash.c file to indicate which extra credit features (EC1, EC2, EC3, and EC4) have been implemented to receive credit. If there is no indication that extra credit features are implemented, no extra credit will be awarded.
>
> Example of **required** comment:
>
> `// EXTRA CREDIT FEATURES: EC2, EC3 implemented`

- Helps graders identify if they should evaluate extra credit
- If comments are missing from **assignment #1**, please go to assignment #1, click:
  "**Submission Details**" link on the RIGHT
- Add a comment in the "**Add a comment**" box
- Indicate which extra credit (EC1, EC2, EC3, EC4) needs graded

| May 28, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.6 |

## OBJECTIVES – 5/28

- Questions from 5/26
- Tutorial 2 (pthreads, locks, conditions) – due Thurs June 4
- Quiz 3 posted – Active Reading Ch. 19 – due Tues June 2
- Assignment 2 (based on Ch. 30) – due Sun *May 31*
- Assignment 3 – on Linux kernel programming – to be offered in "tutorial" format – to be posted ~May 28
- Chapter 20: Paging: Smaller Tables
  - Smaller Tables, Hybrid Tables, Multi-level Page Tables
- Chapter 21/22: Beyond Physical Memory
  - Swapping Mechanisms
  - Swapping Policies
- Chapter 36/37: I/O Devices, Hard Disk Drives

| May 28, 2020 | TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma | L17.7 |

## OBJECTIVES – 5/28

- Questions from 5/26
- Tutorial 2 (pthreads, locks, conditions) – due Thurs June 4
- Quiz 3 posted – Active Reading Ch. 19 – due Tues June 2
- Assignment 2 (based on Ch. 30) – due Sun *May 31*
- Assignment 3 – on Linux kernel programming – to be offered in "tutorial" format – to be posted ~May 28
- Chapter 20: Paging: Smaller Tables
  - Smaller Tables, Hybrid Tables, Multi-level Page Tables
- Chapter 21/22: Beyond Physical Memory
  - Swapping Mechanisms
  - Swapping Policies
- Chapter 36/37: I/O Devices, Hard Disk Drives

| May 28, 2020 | TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma | L17.8 |

## OBJECTIVES – 5/28

- Questions from 5/26
- Tutorial 2 (pthreads, locks, conditions) – due Thurs June 4
- Quiz 3 posted – Active Reading Ch. 19 – due Tues June 2
- Assignment 2 (based on Ch. 30) – due Sun *May 31*
- Assignment 3 – on Linux kernel programming – to be offered in "tutorial" format – to be posted ~May 28
- Chapter 20: Paging: Smaller Tables
  - Smaller Tables, Hybrid Tables, Multi-level Page Tables
- Chapter 21/22: Beyond Physical Memory
  - Swapping Mechanisms
  - Swapping Policies
- Chapter 36/37: I/O Devices, Hard Disk Drives

| May 28, 2020 | TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma | L17.9 |

## OBJECTIVES – 5/28

- Questions from 5/26
- Tutorial 2 (pthreads, locks, conditions) – due Thurs June 4
- Quiz 3 posted – Active Reading Ch. 19 – due Tues June 2
- Assignment 2 (based on Ch. 30) – due *Sun May 28*
- Assignment 3 – on Linux kernel programming – to be offered in "tutorial" format – to be posted ~May 28
- Chapter 20: Paging: Smaller Tables
  - Smaller Tables, Hybrid Tables, Multi-level Page Tables
- Chapter 21/22: Beyond Physical Memory
  - Swapping Mechanisms
  - Swapping Policies
- Chapter 36/37: I/O Devices, Hard Disk Drives

| May 28, 2020 | TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma | L17.10 |

# CHAPTER 20: PAGING: SMALLER TABLES

| May 28, 2020 | TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma | L17.11 |

## OBJECTIVES – 5/28

- Questions from 5/26
- Tutorial 2 (pthreads, locks, conditions) – due Thurs June 4
- Quiz 3 posted – Active Reading Ch. 19 – due Tues June 2
- Assignment 2 (based on Ch. 30) – due …… *Sun May 31*
- Assignment 3 – on Linux kernel programming – to be offered in "tutorial" format – to be posted ~May 28
- Chapter 20: Paging: Smaller Tables
  - Smaller Tables, Hybrid Tables, Multi-level Page Tables
- Chapter 21/22: Beyond Physical Memory
  - Swapping Mechanisms
  - Swapping Policies
- Chapter 36/37: I/O Devices, Hard Disk Drives

| May 28, 2020 | TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma | L17.12 |

## Slide L17.13

### MORE THAN TWO LEVELS

- Consider: Address space of 1 GB
- Page size is $2^9$ = 512 bytes
- Page size 512 bytes / Page entry size 4 bytes
- VPN is 21 bits



| Flag | Detail |
|------|--------|
| Virtual address | 30 bit |
| Page size | 512 byte |
| VPN | 21 bit |
| Offset | 9 bit |

| May 28, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.13 |

## Slide L17.14

### MORE THAN TWO LEVELS - 2

- Page table entries per page = 512 / 4 = 128
- **SPLIT 21 bit VPN:** 7 bits – for page table index (PTI)



| Flag | Detail | |
|------|--------|--|
| Virtual address | 30 bit | |
| Page size | 512 byte | |
| VPN | 21 bit | |
| Offset | 9 bit | |
| Page entry per page | 128 PTEs | $\log_2 128 = 7$ |

| May 28, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.14 |

## Slide L17.15

### MORE THAN TWO LEVELS - 3

- To map 1 GB address space ($2^{30}$=1GB RAM, 512-byte pages)
- $2^{14}$ = 16,384 page directory entries (PDEs) are required
- When using $2^7$ (128 entry) page tables…
- Page size = 128 entries x 4 bytes per addr = 512 bytes



| Flag | Detail | |
|------|--------|--|
| Virtual address | 30 bit | |
| Page size | 512 byte | |
| VPN | 21 bit | |
| Offset | 9 bit | |
| Page entry per page | 128 PTEs | $\log_2 128 = 7$ |

| May 28, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.15 |

## Slide L17.16

### MORE THAN TWO LEVELS - 3

- To map 1 GB address space ($2^{30}$=1GB RAM, 512-byte pages)
- $2^{14}$ = 16,384 page directory entries (PDEs) are required
- When using $2^7$ (128 entry) page tables…
- Page size = 128 entries x 4 bytes per addr = 512 bytes

> Can't Store Page Directory with 16K
> pages, using 512 bytes pages.
> Pages only fit (dereference)
> 128 addresses = (512 bytes / 4 bytes)

| Virtual address | 30 bit | |
|------|--------|--|
| Page size | 512 byte | |
| VPN | 21 bit | |
| Offset | 9 bit | |
| Page entry per page | 128 PTEs | $\log_2 128 = 7$ |

| May 28, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.16 |

## Slide L17.17

### MORE THAN TWO LEVELS - 3

- To map 1 GB address space ($2^{30}$=1GB RAM, 512-byte pages)
- $2^{14}$ = 16,384 page directory entries (PDEs) are required
- When using $2^7$ (128 entry) page tables…
- Page size = 128 entries x 4 bytes per addr = 512 bytes

> Need three level page table:
> Page directory 0 (PD Index 0- 7bit)
> Page directory 1 (PD Index 1- 7bit)
> Small Page Table (Page Table Index- 7bit)

| Virtual address | 30 bit | |
|------|--------|--|
| Page size | 512 byte | |
| VPN | 21 bit | |
| Offset | 9 bit | |
| Page entry per page | 128 PTEs | $\log_2 128 = 7$ |

| May 28, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.17 |

## Slide L17.18

### MORE THAN TWO LEVELS - 4

- We can now address 1GB with "fine grained" 512 byte pages
- Using multiple levels of indirection



- Consider the implications for address translation!
- How much space is required for a virtual address space with only 4 entries on a 512-byte page? (e.g. 4 x 32-bit integers)
- PD0 1 page, PD1 1 page, PT 1 page = 1,536 bytes
- Memory Usage= 1,536 (*3-level*) / 8,388,608 (*1-level*) = .0183% !!!

| May 28, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.18 |

## ADDRESS TRANSLATION CODE

```
// 5-level Linux page table address lookup
//
// Inputs:
// mm_struct – process's memory map struct
// vpage – virtual page address

// Define page struct pointers
pgd_t *pgd;
p4d_t *p4d;
pud_t *pud;
pmd_t *pmt;
pte_t *pte;
struct page *page;
```

May 28, 2020 | TCSS422: Operating Systems [Spring 2020]
School of Engineering and Technology, University of Washington - Tacoma | L17.19

## ADDRESS TRANSLATION - 2

```
pgd = pgd_offset(mm, vpage);
if (pgd_none(*pgd) || pgd_bad(*pgd))
    return 0;
p4d = p4d_offset(pgd, vpage);
if (p4d_none(*p4d) || p4d_bad(*p4d))
    return 0;
pud = pud_offset(p4d, vpage);
if (pud_none(*pud) || pud_bad(*pud))
    return 0;
pmd = pmd_offset(pud, vpage);
if (pmd_none(*pmd) || pmd_bad(*pmd))
    return 0;
if (!(pte = pte_offset_map(pmd, vpage)))
    return 0;
if (!(page = pte_page(*pte)))
    return 0;
physical_page_addr = page_to_phys(page);
pte_unmap(pte);
return physical_page_addr;  // param to send back
```

**pgd_offset():**
Takes a vpage address and the mm_struct for the process, returns the PGD entry that covers the requested address...

**p4d/pud/pmd_offset():**
Takes a vpage address and the pgd/p4d/pud entry and returns the relevant p4d/pud/pmd.

**pte_unmap()**
release temporary kernel mapping for the page table entry

May 28, 2020 | TCSS422: Operating Systems [Spring 2020]
School of Engineering and Technology, University of Washington - Tacoma | L17.20

## INVERTED PAGE TABLES

- Keep a single page table for each physical page of memory

- Consider 4GB physical memory
- Using 4KB pages, page table requires 4MB to map all of RAM

- Page table stores
  - Which process uses each page
  - Which process virtual page (from process virtual address space) maps to the physical page

- All processes share the same page table for memory mapping, kernel must isolate all use of the shared structure
- Finding process memory pages requires search of $2^{20}$ pages
- Hash table: can index memory and speed lookups

May 28, 2020 | TCSS422: Operating Systems [Spring 2020]
School of Engineering and Technology, University of Washington - Tacoma | L17.21

## (#1) Consider a 16 MB Address Space ($2^{24}$) which is indexed using 4KB pages. For a single-level page table, how many pages are required to index memory?

$2^8$ pages

$2^{10}$ pages

$2^{12}$ pages

$2^{14}$ pages

$2^{16}$ pages

May 28, 2020   Start the presentation to see polls in help at PollEv.com/app

School of Engineering and Technology, University of Washington - Tacoma | L17.2

## (#2) For this 16 MB Address Space ($2^{24}$) indexed using 4KB pages, how many bits are required for the VPN?

8 bits

16 bits

10 bits

14 bits

12 bits

May 28, 2020   Start the presentation to see polls in help at PollEv.com/app

School of Engineering and Technology, University of Washington - Tacoma | L17.3

## (#3) Assuming 4 KB pages, how many offset bits are required to index any byte on the page?

6 bits

10 bits

8 bits

12 bits

14 bits

May 28, 2020   Start the presentation to see polls in help at PollEv.com/app

School of Engineering and Technology, University of Washington - Tacoma | L17.4

**(#4) Assuming there are 20 status bits, how many bytes are required for each page table entry (PTE)?**

1 byte

2 bytes

3 bytes

4 bytes

0 bytes

**(#5) How many kilobytes (KB) are required for a single level page table?**

32 KB

16 KB

64 KB

8 KB

24 KB

**(#6) For HelloWorld.c with 4 memory pages: 1 code, stack, heap, data segment, assuming a 2-level page table, how many bits are required for the Page Directory Index (PDI) ?**

6 bits

12 bits

10 bits

8 bits

14 bits

**(#7) For HelloWorld.c with 4 memory pages: 1 code, stack, heap, data segment, assuming a 2-level page table, how many bits are required for the Page Table Index (PTI)**

14 bits

12 bits

10 bits

8 bits

6 bits

**(#8) How much total memory is required to index HelloWorld.c using a two-level page table with just 4 total pages (1 code, stack, heap, data segment page). Hint: need 1 PD and 1 PT**

256 bytes

512 bytes

1024 bytes

2048 bytes

4096 bytes

**(#9) For a 2-level page table, using a single Page Directory Entry (PDE) pointing to a single Page Table (PT), where all slots of the PT are used, how much memory can be addressed?**

16 entries x 4096 bytes = 64 KB

32 entries x 4096 bytes = 128 KB

64 entries x 4096 bytes = 256 KB

256 entries x 4096 bytes = 1024 KB

4096 entries x 4096 bytes = 16384 KB

---

**(#10) For the previous example where one PDE points to a fully used PT, what percentage of memory does the 2-level page table consume vs. a 1-level page table?**

256 / 16384

512 / 16384

1024 / 16384

4096 / 16384

100%

May 28, 2020 TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma L17.1

---

## MULTI-LEVEL PAGE TABLE EXAMPLE

- Consider a 16 MB computer which indexes memory using 4KB pages

- **(#1)** For a single level page table, how many pages are required to index memory?

- **(#2)** How many bits are required for the VPN?

- **(#3)** Assuming each page table entry (PTE) can index any byte on a 4KB page, how many offset bits are required?

- **(#4)** Assuming there are 20 status bits, how many bytes are required for each page table entry?

May 28, 2020 TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma L17.32

---

## MULTI LEVEL PAGE TABLE EXAMPLE - 2

- **(#5)** How many bytes (or KB) are required for a single level page table?

- Let's assume a simple HelloWorld.c program.
- HelloWorld.c requires virtual address translation for 4 pages:
  - 1 – code page
  - 1 – heap page
  - 1 – stack page
  - 1 – data segment page

- **(#6)** Assuming a two-level page table scheme, how many bits are required for the Page Directory Index (PDI)?

- **(#7)** How many bits are required for the Page Table Index (PTI)?

May 28, 2020 TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma L17.33

---

## MULTI LEVEL PAGE TABLE EXAMPLE - 3

- Assume each page directory entry (PDE) and page table entry (PTE) requires 4 bytes:
  - 6 bits for the Page Directory Index (PDI)
  - 6 bits for the Page Table Index (PTI)
  - 12 offset bits
  - 20 status bits

- **(#8)** How much **total** memory is required to index the HelloWorld.c program using a two-level page table when we only need to translate 4 total pages?
- HINT: we need to allocate one Page Directory and one Page Table…
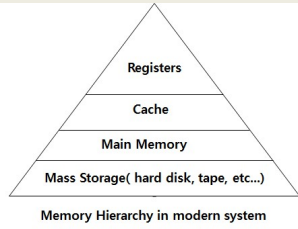- HINT: how many entries are in the PD and PT

May 28, 2020 TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma L17.34

---

## MULTI LEVEL PAGE TABLE EXAMPLE - 4

- **(#9)** Using a single page directory entry (PDE) pointing to a single page table (PT), if all of the slots of the page table (PT) are in use, what is the total amount of memory a two-level page table scheme can address?

- **(#10)** And finally, for this example, as a percentage (%), how much memory does the 2-level page table scheme consume compared to the 1-level scheme?
- HINT: two-level memory use / one-level memory use

May 28, 2020 TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma L17.35

---

## ANSWERS

- #1 – 4096 pages
- #2 – 12 bits
- #3 – 12 bits
- #4 – 4 bytes
- #5 – 4096 x 4 = 16,384 bytes (16KB)
- #6 – 6 bits
- #7 – 6 bits
- #8 – 256 bytes for Page Directory (PD)    (64 entries x 4 bytes)
       256 bytes for Page Table (PT)        **TOTAL = 512 bytes**
- #9 – 64 entries, where each entry maps a 4,096 byte page With 12 offset bits, can address 262,144 bytes (256 KB)
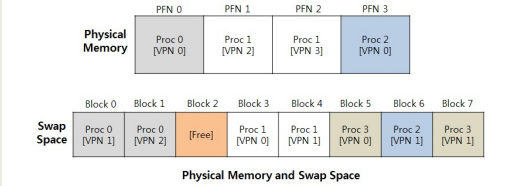- #10- 512/16384 = .03125 → 3.125%

May 28, 2020 TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma L17.36

---

## CHAPTER 21/22: BEYOND PHYSICAL MEMORY

---

## OBJECTIVES – 5/28

- Questions from 5/26
- Tutorial 2 (pthreads, locks, conditions) – due Thurs June 4
- Quiz 3 posted – Active Reading Ch. 19 – due Tues June 2
- Assignment 2 (based on Ch. 30) – due Sun *May 31*
- Assignment 3 – on Linux kernel programming –
  to be offered in "tutorial" format – to be posted ~May 28
- Chapter 20: Paging: Smaller Tables
  - Smaller Tables, Hybrid Tables, Multi-level Page Tables
- Chapter 21/22: Beyond Physical Memory
  - Swapping Mechanisms
  - Swapping Policies
- Chapter 36/37: I/O Devices, Hard Disk Drives

---

## MEMORY HIERARCHY

- Disks (HDD, SSD) provide another level of storage in the memory hierarchy

Registers

Cache

Main Memory

Mass Storage( hard disk, tape, etc...)

Memory Hierarchy in modern system

---

## MOTIVATION FOR EXPANDING THE ADDRESS SPACE

- Provide the illusion of an address space larger than physical RAM

- For a single process
  - Convenience
  - Ease of use

- For multiple processes
  - Large virtual memory space supports running *many concurrent processes. . .*

---

## LATENCY TIMES

- Design considerations:
  - SSDs 4x the time of DRAM
  - HDDs 80x the time of DRAM

| Action | Latency (ns) | (µs) | |
|---|---|---|---|
| L1 cache reference | 0.5ns | | |
| L2 cache reference | 7 ns | | 14x L1 cache |
| Mutex lock/unlock | 25 ns | | |
| Main memory reference | 100 ns | | 20x L2 cache, 200x L1 |
| Read 4K randomly from SSD* | 150,000 ns | 150 µs | ~1GB/sec SSD |
| Read 1 MB sequentially from memory | 250,000 ns | 250 µs | |
| Read 1 MB sequentially from SSD* | 1,000,000 ns | 1,000 µs | 1 ms ~1GB/sec SSD, 4X memory |
| Read 1 MB sequentially from disk | 20,000,000 ns | 20,000 µs | 20 ms 80x memory, 20X SSD |

- *Latency numbers every programmer should know*
- From: https://gist.github.com/jboner/2841832#file-latency-txt

---

## SWAP SPACE

- Disk space for storing memory pages
- "Swap" them in and out of memory to disk as needed

Physical Memory and Swap Space

## SWAP SPACE - 2

- The size of the swap space can be seen using the Linux free command: "free –h"

```
wlloyd@dione:~$ free -h
              total        used        free      shared  buff/cache   available
Mem:            30G         11G         14G        1.3G        4.4G         17G
Swap:           31G          0B         31G
```

- With sufficient disk space, a common allocation is to create Swap space greater than or equal to physical RAM

## SWAP SPACE - 3

- Swap space lives on a separate logical volume in Ubuntu Linux that is managed separately from the root file system

- Check logical volumes with "sudo lvdisplay" command:

```
--- Logical volume ---
LV Path                /dev/ubuntu-vg/swap_1
LV Name                swap_1
VG Name                ubuntu-vg
LV UUID                G1Ovj6-4M33-2YXY-YETH-wf7V-93vF-QRQytG
LV Write Access        read/write
LV Creation host, time ubuntu, 2018-09-30 15:44:16 -0700
LV Status              available
# open                 2
LV Size                976.00 MiB
Current LE             244
Segments               1
Allocation             inherit
Read ahead sectors     auto
- currently set to     256
Block device           253:1
```

- See also "lvm lvs" command

## PAGE LOCATION

- Memory pages are:
  - Stored in memory
  - Swapped to disk

- Present bit
  - In the page table entry (PTE) indicates if page is present

- Page fault
  - Memory page is accessed, but has been swapped to disk

## PAGE FAULT

- OS steps in to handle the page fault

- Loading page from disk requires a free memory page

- Page-Fault Algorithm

```
1:    PFN = FindFreePhysicalPage()
2:    if (PFN == -1)                    // no free page found
3:        PFN = EvictPage()             // run replacement algorithm
4:    DiskRead(PTE.DiskAddr, pfn)       // sleep (waiting for I/O)
5:    PTE.present = True                // set PTE bit to present
6:    PTE.PFN = PFN                     // reference new loaded page
7:    RetryInstruction()                // retry instruction
```

## PAGE REPLACEMENTS

- Page daemon
  - Background threads which monitors swapped pages

- Low watermark (LW)
  - Threshold for when to swap pages to disk
  - Daemon checks: free pages < LW
  - Begin swapping to disk until reaching the highwater mark

- High watermark (HW)
  - Target threshold of free memory pages
  - Daemon free until: free pages >= HW

## TCSS 422 WILL RETURN AT ~2:40PM

## REPLACEMENT POLICIES

POLICY CHANGES

---

## CACHE MANAGEMENT

- Replacement policies apply to "any" cache
- Goal is to minimize the number of misses
- 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

---

## OPTIMAL REPLACEMENT POLICY

- What if:
  - We could predict the future (... with a magical oracle)
  - All future page accesses are known
  - Always replace the page in the cache used farthest in the future

- Used for a comparison
- Provides a "best case" replacement policy

- Consider a 3-element empty cache with the following page accesses:

  0 1 2 0 1 3 0 3 1 2 1

  **What is the hit/miss ratio?**

  **6 hits**

---

## FIFO REPLACEMENT

- Queue based
- **Always replace the oldest element** at the back of cache
- Simple to implement
- Doesn't consider importance... just arrival ordering

- Consider a 3-element empty cache with the following page accesses:
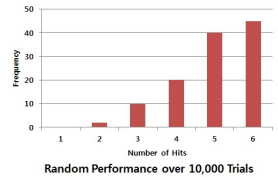
  0 1 2 0 1 3 0 3 1 2 1

- What is the hit/miss ratio?   **4 hits**
- How is FIFO different than LRU?   **LRU incorporates history**

---

## RANDOM REPLACEMENT

- Pick a page at random to replace
- Simple and fast implementation
- Performance depends on luck of random choices

  0 1 2 0 1 3 0 3 1 2 1

Random Performance over 10,000 Trials

---

## HISTORY-BASED POLICIES

- LRU: Least recently used
- Always replace page with oldest access time (front)
- Always move end of cache when element is read again
- LRU requires constant reorganization of the cache
- Considers temporal locality (*when pg was last accessed*)

  0 1 2 0 1 3 0 3 1 2 1   **What is the hit/miss ratio?**

  **6 hits**

- LFU: Least frequently used
- Always replace page with the fewest # of accesses (front)
- Incorporates frequency of use - *must track pg accesses*
- Consider frequency of page accesses

  0 1 2 0 1 3 0 3 1 2 1   **Hit/miss ratio is=6 hits**

---

**Consider a 3-element cache. With a FIFO replacement policy, how many hits occur with the following page access sequence:**

1 2 0 1 3 1 2 0 2 1 3

2 hits

3 hits

4 hits

5 hits

6 hits

May 28, 2020    TCSS422: Operating Systems [Spring 2020]    L17.5
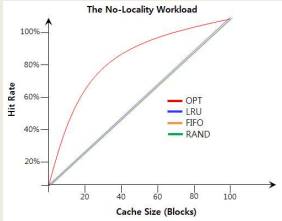School of Engineering and Technology, University of Washington - Tacoma

---

**Consider a 3-element cache. With an LRU replacement policy, how many hits occur with the following page access sequence:**

1 2 0 1 3 1 2 0 2 1 3

2 hits

3 hits

4 hits

5 hits

6 hits

May 28, 2020    TCSS422: Operating Systems [Spring 2020]    L17.6
School of Engineering and Technology, University of Washington - Tacoma

---

## WORKLOAD EXAMPLES: NO-LOCALITY

- No-Locality (Random Access) Workload
  - Perform 10,000 random page accesses
  - Across set of 100 memory pages
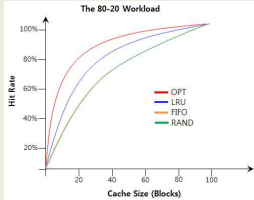
When the cache is large enough to fit the entire workload, it doesn't matter which policy you use.

May 28, 2020    TCSS422: Operating Systems [Spring 2020]    L17.57
School of Engineering and Technology, University of Washington - Tacoma

---

## WORKLOAD EXAMPLES: 80/20

- 80/20 Workload
  - Perform 10,000 page accesses, against set of 100 pages
  - 80% of accesses are to 20% of pages (hot pages)
  - 20% of accesses are to 80% of pages (cold pages)

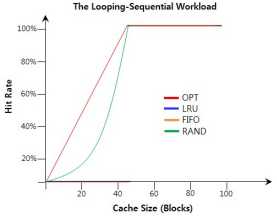LRU is more likely to hold onto hot pages

(recalls history)

May 28, 2020    TCSS422: Operating Systems [Spring 2020]    L17.58
School of Engineering and Technology, University of Washington - Tacoma

---

## WORKLOAD EXAMPLES: SEQUENTIAL

- Looping sequential workload
  - Refer to 50 pages in sequence: 0, 1, …, 49
  - Repeat loop

Random performs better than FIFO and LRU for cache sizes < 50
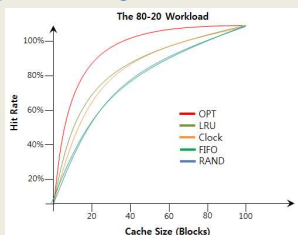
Algorithms should provide "scan resistance"

May 28, 2020    TCSS422: Operating Systems [Spring 2020]    L17.59
School of Engineering and Technology, University of Washington - Tacoma

---

**With small cache sizes, for the looping sequential workload, why do FIFO and LRU fail to provide cache hits?**

Cache hits in this scenario require consideration of how frequently accessed memory is for cache replacement

Memory accesses are unpredictable and too random. Unpredictable accesses require a random cache replacement policy for cache hits

Memory accesses to elements that are accessed repeatedly are too spread apart temporally to benefit from caching

Unlike Random cache replacement, both FIFO and LRU fail to speculate memory accesses in advance to improve caching

None of the above

Start the presentation to see live content. For screen share software, share the entire screen. Get help at pollev.com/app

## IMPLEMENTING LRU

- Implementing last recently used (LRU) requires tracking access time for all system memory pages
- Times can be tracked with a list
- For cache eviction, we must scan an entire list
- Consider: 4GB memory system ($2^{32}$), with 4KB pages ($2^{12}$)

- This requires $2^{20}$ comparisons !!!

- Simplification is needed
  - Consider how to approximate the oldest page access

| May 28, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.61 |

## IMPLEMENTING LRU - 2

- Harness the Page Table Entry (PTE) Use Bit
- HW sets to 1 when page is used
- OS sets to 0

- Clock algorithm (*approximate LRU*)
  - Refer to pages in a circular list
  - Clock hand points to current page
  - Loops around
    - IF USE_BIT=1 set to USE_BIT = 0
    - IF USE_BIT=0 replace page

| May 28, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.62 |

## CLOCK ALGORITHM

- Not as efficient as LRU, but better than other replacement algorithms that do not consider history

The 80-20 Workload

OPT
LRU
Clock
FIFO
RAND

| May 28, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.63 |

## CLOCK ALGORITHM - 2

- Consider dirty pages in cache
- If DIRTY (modified) bit is FALSE
  - No cost to evict page from cache

- If DIRTY (modified) bit is TRUE
  - Cache eviction requires updating memory
  - Contents have changed

- Clock algorithm should favor no cost eviction

| May 28, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.64 |

## WHEN TO LOAD PAGES

- On demand → demand paging

- Prefetching
  - Preload pages based on anticipated demand

  - Prediction based on locality
  - Access page P, suggest page P+1 may be used

- What other techniques might help anticipate required memory pages?
  - Prediction models, historical analysis
  - In general: accuracy vs. effort tradeoff
  - High analysis techniques struggle to respond in real time

| May 28, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.65 |

## OTHER SWAPPING POLICIES

- Page swaps / writes
  - Group/cluster pages together
  - Collect pending writes, perform as batch
  - Grouping disk writes helps amortize latency costs

- Thrashing
  - Occurs when system runs many memory intensive processes and is low in memory
  - Everything is constantly swapped to-and-from disk

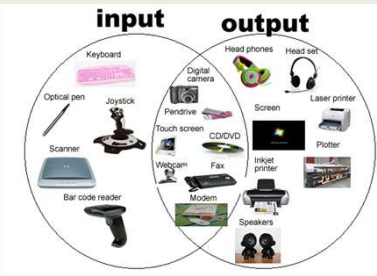| May 28, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.66 |

## OTHER SWAPPING POLICIES - 2

- Working sets
  - Groups of related processes
  - When thrashing: prevent one or more working set(s) from running
  - Temporarily reduces memory burden
  - Allows some processes to run, reduces thrashing

May 28, 2020 — TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma — L17.67

## CHAPTER 36: I/O DEVICES

May 26, 2020 — TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma — L17.68

## OBJECTIVES – 5/28

- Questions from 5/26
- Tutorial 2 (pthreads, locks, conditions) – due Thurs June 4
- Quiz 3 posted – Active Reading Ch. 19 – due Tues June 2
- Assignment 2 (based on Ch. 30) – due Sun *May 31*
- Assignment 3 – on Linux kernel programming – to be offered in "tutorial" format – to be posted ~May 28
- Chapter 20: Paging: Smaller Tables
  - Smaller Tables, Hybrid Tables, Multi-level Page Tables
- Chapter 21/22: Beyond Physical Memory
  - Swapping Mechanisms
  - Swapping Policies
- Chapter 36/37: I/O Devices, Hard Disk Drives

May 28, 2020 — TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma — L17.69

## OBJECTIVES

- Chapter 36

  - Polling vs Interrupts

  - Programmed I/O (PIO)
    - Port-mapped I/O (PMIO)
    - Memory-mapped I/O (MMIO)

  - Direct memory Access (DMA)

May 26, 2020 — TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma — L17.70

## I/O DEVICES

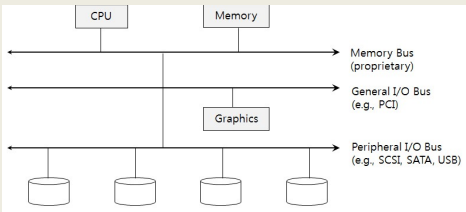- Modern computer systems interact with a variety of devices

May 26, 2020 — TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma — L17.71

## 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.

May 26, 2020 — TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma — L17.72

## 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

| May 26, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.73 |

## CANONICAL DEVICE

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

| Registers: | Status | Command | Data | interface |
| --- | --- | --- | --- | --- |
| Micro-controller(CPU)<br>Memory (DRAM or SRAM or both)<br>Other Hardware-specific Chips | | | | internals |

**Canonical Device**

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

| May 26, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.74 |

## 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.

| May 26, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.75 |

## OS DEVICE INTERACTION

- Common example of device interaction

```
while ( STATUS == BUSY)          Poll- Is device available?
  ; //wait until device is not busy
write data to data register      Command parameterization
write command to command register   Send command
  Doing so starts the device and executes the command
while ( STATUS == BUSY)          Poll – Is device done?
  ; //wait until device is done with your request
```

| May 26, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.76 |

## 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

"waiting IO"      | 1 | : task 1  | P | : polling

| CPU | 1 | 1 | 1 | 1 | 1 | P | P | P | P | P | 1 | 1 | 1 | 1 | 1 |

| Disk | 1 | 1 | 1 | 1 | 1 |

**CPU utilization by polling**

| May 26, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.77 |

## 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)

| 1 | : task 1  | 2 | : task 2

| CPU | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 |

| Disk | 1 | 1 | 1 | 1 | 1 |

**Diagram of CPU utilization by interrupt**

| May 26, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.78 |

## 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.
  If device I/O is slow → interrupts are better.

| May 26, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.79 |

## INTERRUPTS VS POLLING - 3

- One solution is a two-phase hybrid approach
  - Initially poll, then sleep and use interrupts

- 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

| May 26, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.80 |

## DEVICE I/O

- To interact with a device we must send/receive DATA

- There are two general approaches:

  - Programmed I/O (PIO)

  - Direct memory access (DMA)

| May 26, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.81 |

| 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

## PROGRAMMED I/O (PIO)

- Spend CPU time to perform I/O
- CPU is involved with the data movement (input/output)
- PIO is slow – CPU is occupied with meaningless work



Diagram of CPU utilization

| May 26, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.83 |

## PIO DEVICES

- Legacy serial ports

- Legacy parallel ports

- PS/2 keyboard and mouse

- Legacy MIDI, joysticks

- Old network interfaces

| May 26, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.84 |

## 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 CISC model: extra instructions

- 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 CPU memory
- Tenet of RISC CPUs: instructions are eliminated, CPU is simpler
- Old days: 16-bit CPUs didn't have a lot of spare memory space
- Today's CPUs: 32-bit (4GB addr space) & 64-bit (128 TB addr space)
- Regular CPU instructions used to access device: mapped to memory
- Devices monitor CPU address bus and respond to their addresses
- I/O device address areas of memory are **reserved** for I/O
  - Must not be available for normal memory operations.

## 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: for DISK I/O consider the variety of disks:

- SCSI, IDE, USB flash drive, DVD, etc.

- Device drivers use abstraction to provide general interfaces for vendor specific hardware

- In Linux: block devices

## FILE SYSTEM ABSTRACTION

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

| Application | user |
| --- | --- |
| POSIX API [open, read, write, close, etc] | |
| File System | kernel |
| Generic Block Interface [block read/write] | |
| Generic Block Layer | |
| Specific Block Interface [protocol-specific read/write] | |
| Device Driver [SCSI, ATA, etc] | |

**The File System Stack**

May 26, 2020 | TCSS422: Operating Systems [Spring 2020]
School of Engineering and Technology, University of Washington - Tacoma | L17.91

## FILE SYSTEM ABSTRACTION ISSUES

- **Too much abstraction**

- Many devices provide special capabilities
- Example: SCSI Error handling
- SCSI devices provide extra detail which are lost to the OS

- **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.)

May 26, 2020 | TCSS422: Operating Systems [Spring 2020]
School of Engineering and Technology, University of Washington - Tacoma | L17.92

## CH. 37: HARD DISK DRIVES

May 26, 2020 | TCSS422: Operating Systems [Spring 2020]
School of Engineering and Technology, University of Washington - Tacoma | L17.93

## OBJECTIVES – 5/28

- **Questions from 5/26**
- **Tutorial 2 (pthreads, locks, conditions) – due Thurs June 4**
- **Quiz 3 posted – Active Reading Ch. 19 – due Tues June 2**
- **Assignment 2 (based on Ch. 30) – due Sun *May 31***
- **Assignment 3 – on Linux kernel programming – to be offered in "tutorial" format – to be posted ~May 28**
- **Chapter 20: Paging: Smaller Tables**
  - **Smaller Tables, Hybrid Tables, Multi-level Page Tables**
- **Chapter 21/22: Beyond Physical Memory**
  - **Swapping Mechanisms**
  - **Swapping Policies**
- **Chapter 36, 37: I/O Devices, Hard Disk Drives**

May 28, 2020 | TCSS422: Operating Systems [Spring 2020]
School of Engineering and Technology, University of Washington - Tacoma | L17.94

## OBJECTIVES

- Chapter 37

  - **HDD Internals**

  - **Seek time**

  - **Rotational latency**

  - **Transfer speed**

  - **Capacity**

  - **Scheduling algorithms**

May 26, 2020 | TCSS422: Operating Systems [Spring 2020]
School of Engineering and Technology, University of Washington - Tacoma | L17.95

## HARD DISK DRIVE (HDD)

- **Primary means of data storage (persistence) for decades**

- **Consists of a large number of data sectors**
- **Sector size is 512-bytes**

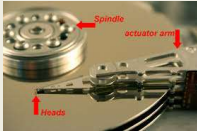- **An n sector HDD can be is addressed as an array of 0..n-1 sectors**

May 26, 2020 | TCSS422: Operating Systems [Spring 2020]
School of Engineering and Technology, University of Washington - Tacoma | L17.96

## 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

Spindle
actuator arm
Heads

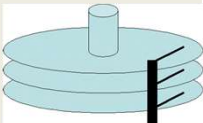| May 26, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.97 |

## BLOCK SIZE IN LINUX EXT4

- mkefs.ext4  -i bytes-per-inode

Specify the bytes/inode ratio. mke2fs creates an inode for every bytes-per-inode bytes of space on the disk. The larger the bytes-per-inode ratio, the fewer inodes will be created. This value generally shouldn't be smaller than the blocksize of the filesystem, since in that case more inodes would be made than can ever be used. Be warned that it is not possible to expand the number of inodes on a filesystem after it is created, so be careful deciding the correct value for this parameter.

| May 26, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.98 |

## EXAMPLE: USDA SOIL EROSION MODEL WEB SERVICE (RUSLE2)

- Host ~2,000,000 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

| May 26, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.99 |

## 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 |

| May 26, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.100 |

## HDD INTERFACE - 2

- Torn write
  - When OS uses larger block size than HDD
  - Block writes not atomic - they SPAN multiple HDD sectores
  - Upon power failure only a portion of the OS block is written

- HDD access
  - Sequential reads of sectors is fastest
  - Random sector reads are slow
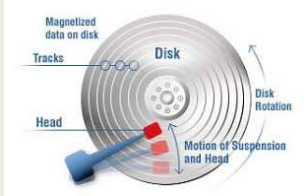  - Disk head continuously must jump to different tracks

| May 26, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.101 |

## HDD PLATTER

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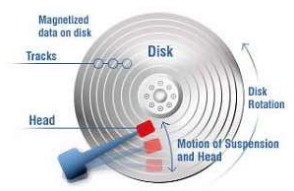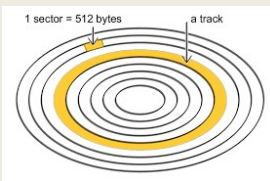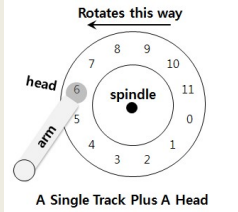- Data is stored by inducing magnetic changes

Magnetized data on disk
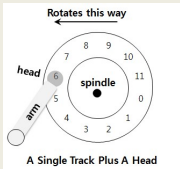Tracks
Disk
Head
Disk Rotation
Motion of Suspension and Head

| May 26, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.102 |

## 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

## 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



1 sector = 512 bytes    a track

## EXAMPLE: SIMPLE DISK DRIVE

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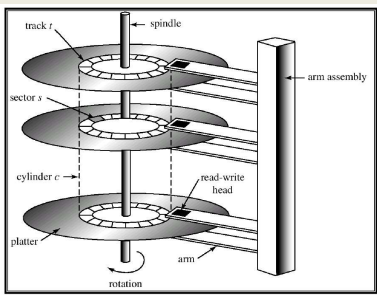


A Single Track Plus A Head

## HARD DISK STRUCTURE

## SINGLE-TRACK LATENCY: THE ROTATIONAL DELAY

- Rotational latency ($T_{rotation}$): time to rotate to desired sector

- Average $T_{rotation}$ is ~ half the time of a full rotation

- Calculate time for 1 rotation based on rpm

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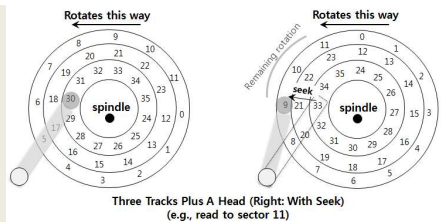- 15000rpm = 4ms per rotation = ~2ms



A Single Track Plus A Head

## 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

## 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

| May 26, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.109 |

## 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)
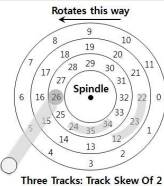  2. Wait on rotational latency
  3. Data transfer

| May 26, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.110 |

## TRACK SKEW

- Sectors are offset across tracks to allow time for head to reposition for sequential reads

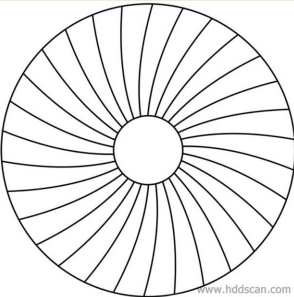- Without track skew, when head is repositioned sector would have already been passed



Three Tracks: Track Skew Of 2

| May 26, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.111 |

## TRACK SKEW - 2



| May 26, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.112 |

## HDD CACHE

- Buffer to support caching reads and writes
- Improves drive response time
- Up to 128 MB, slowly have been growing

- Two styles
  - Writeback cache
    - Report write complete immediately when data is transferred to HDD cache
    - Dangerous
  - Writethrough cache
    - Reports write complete only when write is physically completed on disk

| May 26, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.113 |

## TRANSFER SPEED

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

- The rate of I/O $R_{I/O} = \dfrac{Size_{Transfer}}{T_{I/O}}$

|  | 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

| May 26, 2020 | TCSS422: Operating Systems [Spring 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L17.114 |

## I/O SPEED

- Random workload: 4KB random read on HDD
- Sequential workload: read 100MB contiguous sectors

| | | Cheetah 15K.5 | Barracuda |
|---|---|---|---|
| $T_{seek}$ | | 4 ms | 9 ms |
| $T_{rotation}$ | | 2 ms | 4.2 ms |
| 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 |
| 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**

| May 26, 2020 | TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma | L17.115 |

## MODERN HDD SPECS

- See sample HDD configurations here:

- https://www.hgst.com/products/hard-drives

| May 26, 2020 | TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma | L17.116 |

## 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

| May 26, 2020 | TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma | L17.117 |

## 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



SSTF: Scheduling Request 21 and 2
Issue the request to 21 → issue the request to 2

| May 26, 2020 | TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma | L17.118 |

## 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

| May 26, 2020 | TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma | L17.119 |

## DISK SCHEDULING ALGORITHMS

- **SWEEP**
- Single repeated passes across disk
- Issue: if request arrives for a recently visited track it will not be revisited until a full cycle completes

- **F-SCAN**
- Freeze request queue during sweep
- Cache arriving requests until later

- **Elevator (C-SCAN)** – circular scan
- Sweep from outer to inner track and reverse, inner to outer track, etc.

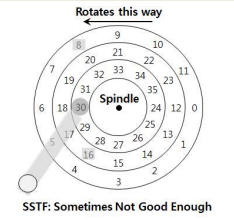| May 26, 2020 | TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma | L17.120 |

## SHORTEST TIME POSITIONING FIRST

- Determine next sector to read?

- On which track?

- On which sector?

Rotates this way

Spindle

SSTF: Sometimes Not Good Enough

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

May 26, 2020 | TCSS422: Operating Systems [Spring 2020]
School of Engineering and Technology, University of Washington - Tacoma | L17.121

## 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?

May 26, 2020 | TCSS422: Operating Systems [Spring 2020]
School of Engineering and Technology, University of Washington - Tacoma | L17.122

# QUESTIONS

# WILL RETURN IN A FEW MINUTES