


TCCS 422: OPERATING SYSTEMS

Multi-Level Page Tables, Beyond Physical Memory

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



June 1, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma

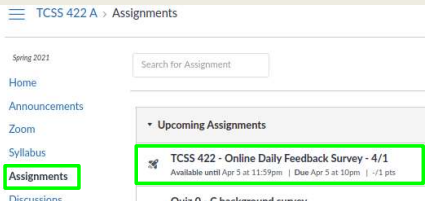
OBJECTIVES – 6/1

- **Questions from 5/27**
- REVIEW: Memory Segmentation Activity (available in Canvas)
- Tutorial 2 – Pthread, locks, conditions tutorial – June 4
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Final exam – alternate format
- Quiz 4 – Page Tables
- Chapter 19: Translation Lookaside Buffer (TLB)
 - Hit-to-Miss Ratios
- Chapter 20: Paging: Smaller Tables
 - Smaller Tables, Multi-level Page Tables, N-level Page Tables
- Chapter 21/22: Beyond Physical Memory
 - Swapping Mechanisms, Swapping Policies

June 1, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma

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, 2021 TCCS422: Computer Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma

TCCS 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

June 1, 2021 TCCS422: Computer Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (51 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.02 (↓ - previous 6.35)**
- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.52 (↑ - previous 5.47)**

June 1, 2021 TCCS422: Computer Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma

FEEDBACK

- **Is It Important to try to increase the hit-to-miss ratio for the TLB when writing a program in C?**
- In our program, one thing we can control is how memory is accessed in arrays or other structures, etc.
- To optimize performance of our programs, we can design our code with the TLB in mind by increasing spatial and temporal locality of memory accesses
 - In some cases specific algorithms require certain memory access patterns. These will be difficult to optimize

June 1, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma

FEEDBACK - 2

- Could you review how we find hits and misses from TLB example? (lecture 17 2up pg19)
 - We will start today by reviewing the TLB (chapter 19)

June 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L18.7
--------------	---	-------

OBJECTIVES – 6/1

- Questions from 5/27
- **Assignment 2 – May 31 (Late Penalty June 2, Closing June 4)**
- Tutorial 2 – Pthread, locks, conditions tutorial – June 4
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Final exam – alternate format
- Quiz 4 – Page Tables
- Chapter 19: Translation Lookaside Buffer (TLB)
 - Hit-to-Miss Ratios
- Chapter 20: Paging: Smaller Tables
 - Smaller Tables, Multi-level Page Tables, N-level Page Tables
- Chapter 21/22: Beyond Physical Memory
 - Swapping Mechanisms, Swapping Policies

June 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L18.8
--------------	---	-------

OBJECTIVES – 6/1

- Questions from 5/27
- Assignment 2 – May 31 (Late Penalty June 2, Closing June 4)
- **Tutorial 2 – Pthread, locks, conditions tutorial – June 4**
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Final exam – alternate format
- Quiz 4 – Page Tables
- Chapter 19: Translation Lookaside Buffer (TLB)
 - Hit-to-Miss Ratios
- Chapter 20: Paging: Smaller Tables
 - Smaller Tables, Multi-level Page Tables, N-level Page Tables
- Chapter 21/22: Beyond Physical Memory
 - Swapping Mechanisms, Swapping Policies

June 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L18.9
--------------	---	-------

OBJECTIVES – 6/1

- Questions from 5/27
- Assignment 2 – May 31 (Late Penalty June 2, Closing June 4)
- Tutorial 2 – Pthread, locks, conditions tutorial – June 4
- **Assignment 3: (Tutorial) Introduction to Linux Kernel Modules**
- Final exam – alternate format
- Quiz 4 – Page Tables
- Chapter 19: Translation Lookaside Buffer (TLB)
 - Hit-to-Miss Ratios
- Chapter 20: Paging: Smaller Tables
 - Smaller Tables, Multi-level Page Tables, N-level Page Tables
- Chapter 21/22: Beyond Physical Memory
 - Swapping Mechanisms, Swapping Policies

June 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L18.10
--------------	---	--------

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

June 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L18.11
--------------	---	--------

OBJECTIVES – 6/1

- Questions from 5/27
- Assignment 2 – May 31 (Late Penalty June 2, Closing June 4)
- Tutorial 2 – Pthread, locks, conditions tutorial – June 4
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- **Final exam – alternate format**
- Quiz 4 – Page Tables
- Chapter 19: Translation Lookaside Buffer (TLB)
 - Hit-to-Miss Ratios
- Chapter 20: Paging: Smaller Tables
 - Smaller Tables, Multi-level Page Tables, N-level Page Tables
- Chapter 21/22: Beyond Physical Memory
 - Swapping Mechanisms, Swapping Policies

June 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L18.12
--------------	---	--------

ALTERNATE FINAL EXAM

- Final Exam category will have two assignments
- Thursday June 10 from 3:40 to 5:40 pm
 - Final Quiz (50 points)
 - SHORT: fewer than half the number of questions as the midterm
 - 1-hour
 - Focus on new content - since the midterm
- Tutorial: Linux File Systems and Disk I/O
 - Available for 1-week ~ June 5th to June 11th
 - 50 points
 - Presents new material in a hands-on, interactive format
 - Complete activity and answer questions
 - Individual work

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.13

OBJECTIVES – 6/1

- Questions from 5/27
- Assignment 2 – May 31 (*Late Penalty June 2, Closing June 4*)
- Tutorial 2 – Pthread, locks, conditions tutorial – June 4
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Final exam – alternate format
- Quiz 4 – Page Tables**
- Chapter 19: Translation Lookaside Buffer (TLB)
 - Hit-to-Miss Ratios
- Chapter 20: Paging: Smaller Tables
 - Smaller Tables, Multi-level Page Tables, N-level Page Tables
- Chapter 21/22: Beyond Physical Memory
 - Swapping Mechanisms, Swapping Policies


June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.14

OBJECTIVES – 6/1

- Questions from 5/27
- Assignment 2 – May 31 (*Late Penalty June 2, Closing June 4*)
- Tutorial 2 – Pthread, locks, conditions tutorial – June 4
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Final exam – alternate format
- Quiz 4 – Page Tables
- Chapter 19: Translation Lookaside Buffer (TLB)
 - Hit-to-Miss Ratios**
- Chapter 20: Paging: Smaller Tables
 - Smaller Tables, Multi-level Page Tables, N-level Page Tables
- Chapter 21/22: Beyond Physical Memory
 - Swapping Mechanisms, Swapping Policies

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.15

CHAPTER 19: TRANSLATION LOOKASIDE BUFFER (TLB)



June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.16

TLB EXAMPLE - 2

```

0: int sum = 0 ;
1: for( i=0; i<10; i++){
2:     sum+=a[i];
3: }
    
```

- Consider the code above:
- Initially the TLB does not know where `a[]` is
- Consider the accesses:
- `a[0], a[1], a[2], a[3], a[4], a[5], a[6], a[7], a[8], a[9]`
- How many pages are accessed?
- What happens when accessing a page not in the TLB?

VPN	00	04	08	12	16
VPN - 00					
VPN - 01					
VPN - 02					
VPN - 03					
VPN - 04					
VPN - 05					
VPN - 06		a[0]	a[1]	a[2]	
VPN - 07	a[3]	a[4]	a[5]	a[6]	
VPN - 08	a[7]	a[8]	a[9]		
VPN - 09					
VPN - 10					
VPN - 11					
VPN - 12					
VPN - 13					
VPN - 14					
VPN - 15					

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.17

TLB EXAMPLE - 3

```

0: int sum = 0 ;
1: for( i=0; i<10; i++){
2:     sum+=a[i];
3: }
    
```

- For the accesses: `a[0], a[1], a[2], a[3], a[4], a[5], a[6], a[7], a[8], a[9]`
- How many are hits?
- How many are misses?
- What is the hit rate? (%)
 - 70% (3 misses one for each VP, 7 hits)

VPN	00	04	08	12	16
VPN - 00					
VPN - 01					
VPN - 02					
VPN - 03					
VPN - 04					
VPN - 05					
VPN - 06		a[0]	a[1]	a[2]	
VPN - 07	a[3]	a[4]	a[5]	a[6]	
VPN - 08	a[7]	a[8]	a[9]		
VPN - 09					
VPN - 10					
VPN - 11					
VPN - 12					
VPN - 13					
VPN - 14					
VPN - 15					

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.18

TLB EXAMPLE - 4

```

0:  int sum = 0 ;
1:  for( i=0; i<10; i++){
2:      sum+=a[i];
3:  }
    
```

- What factors affect the hit/miss rate?
 - Page size
 - Data/Access locality (how is data accessed?)
 - Sequential array access vs. random array access
 - Temporal locality
 - Size of the TLB cache (how much history can you store?)

	OFFSET				
	00	04	08	12	16
VPN - 00					
VPN - 01					
VPN - 02					
VPN - 03					
VPN - 04					
VPN - 05					
VPN - 06					
VPN - 07	a[0]	a[1]	a[2]		
VPN - 08	a[3]	a[4]	a[5]	a[6]	
VPN - 09	a[7]	a[8]	a[9]		
VPN - 10					
VPN - 11					
VPN - 12					
VPN - 13					
VPN - 14					
VPN - 15					


June 1, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L18.19

OBJECTIVES – 6/1

- Questions from 5/27
- Assignment 2 – May 31 (*Late Penalty June 2, Closing June 4*)
- Tutorial 2 – Pthread, locks, conditions tutorial – June 4
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Final exam – alternate format
- Quiz 4 – Page Tables
- Chapter 19: Translation Lookaside Buffer (TLB)
 - Hit-to-Miss Ratios
- Chapter 20: Paging: Smaller Tables
 - Smaller Tables, Multi-level Page Tables, N-level Page Tables
- Chapter 21/22: Beyond Physical Memory
 - Swapping Mechanisms, Swapping Policies

June 1, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L18.20

CHAPTER 20: PAGING: SMALLER TABLES



June 1, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L18.21

LINEAR PAGE TABLES

- Consider array-based page tables:
 - Each process has its own page table
 - 32-bit process address space (up to 4GB)
 - With 4 KB pages
 - 20 bits for VPN
 - 12 bits for the page offset

June 1, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L18.22

LINEAR PAGE TABLES - 2

- Page tables stored in RAM
- Support potential storage of 2²⁰ translations
= 1,048,576 pages per process @ 4 bytes/page
- Page table size 4MB / process

Page table size = $\frac{2^{32}}{2^{12}} * 4Byte = 4MByte$

- Consider 100+ OS processes
 - Requires 400+ MB of RAM to store process information

June 1, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L18.23

LINEAR PAGE TABLES - 2

- Page tables stored in RAM
- Support potential storage of 2²⁰ translations
= 1,048,576 pages per process @ 4 bytes/page
- Page table size 4MB / process

Page tables are too big and consume too much memory.
Need Solutions ...

- Consider 100+ OS processes
 - Requires 400+ MB of RAM to store process information

June 1, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L18.24

OBJECTIVES – 6/1

- Questions from 5/27
- Assignment 2 – May 31 (*Late Penalty June 2, Closing June 4*)
- Tutorial 2 – Pthread, locks, conditions tutorial – June 4
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Final exam – alternate format
- Quiz 4 – Page Tables
- Chapter 19: Translation Lookaside Buffer (TLB)
 - Hit-to-Miss Ratios
- Chapter 20: Paging: Smaller Tables
 - Smaller Tables, Multi-level Page Tables, N-level Page Tables
- Chapter 21/22: Beyond Physical Memory
 - Swapping Mechanisms, Swapping Policies

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.25

PAGING: USE LARGER PAGES

- Larger pages = 16KB = 2^{14}
- 32-bit address space: 2^{32}
- $2^{18} = 262,144$ pages

$$\frac{2^{32}}{2^{14}} * 4 = 1MB \text{ per page table}$$

- Memory requirement cut to $\frac{1}{4}$
- However pages are huge
- Internal fragmentation results
- 16KB page(s) allocated for small programs with only a few variables

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.26

PAGE TABLES: WASTED SPACE

- Process: 16KB Address Space w/ 1KB pages

A 16KB Address Space with 1KB Pages

PFN	valid	prot	present	dirty
10	1	r-x	1	0
-	0	-	-	-
-	0	-	-	-
-	0	-	-	-
15	1	rw-	1	1
...
-	0	-	-	-
3	1	rw-	1	1
23	1	rw-	1	1

A Page Table For 16KB Address Space

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.27

PAGE TABLES: WASTED SPACE

- Process: 16KB Address Space w/ 1KB pages

PFN	valid	prot	present	dirty
10	1	r-x	1	0
-	0	-	-	-
-	0	-	-	-
-	0	-	-	-
15	1	rw-	1	1
...
-	0	-	-	-
3	1	rw-	1	1
23	1	rw-	1	1

A Page Table For 16KB Address Space

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.28

OBJECTIVES – 6/1

- Questions from 5/27
- Assignment 2 – May 31 (*Late Penalty June 2, Closing June 4*)
- Tutorial 2 – Pthread, locks, conditions tutorial – June 4
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Final exam – alternate format
- Quiz 4 – Page Tables
- Chapter 19: Translation Lookaside Buffer (TLB)
 - Hit-to-Miss Ratios
- Chapter 20: Paging: Smaller Tables
 - Smaller Tables, Multi-level Page Tables, N-level Page Tables
- Chapter 21/22: Beyond Physical Memory
 - Swapping Mechanisms, Swapping Policies

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.29

MULTI-LEVEL PAGE TABLES

- Consider a page table:
 - 32-bit addressing, 4KB pages
 - 2^{20} page table entries
 - Even if memory is sparsely populated the *per process* page table requires:

$$\text{Page table size} = \frac{2^{32}}{2^{12}} * 4\text{Byte} = 4\text{MByte}$$

- Often most of the 4MB *per process* page table is empty
- Page table must be placed in 4MB contiguous block of RAM
- MUST SAVE MEMORY!

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.30

MULTI-LEVEL PAGE TABLES - 2

■ Add level of indirection, the "page directory"

Linear (Left) And Multi-Level (Right) Page Tables

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.31

MULTI-LEVEL PAGE TABLES - 2

■ Add level of indirection, the "page directory"

Two level page table:
 2^{20} pages addressed with
 two level-indexing
 (page directory index, page table index)

Linear (Left) And Multi-Level (Right) Page Tables

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.32

MULTI-LEVEL PAGE TABLES - 3

■ Advantages

- Only allocates page table space in proportion to the address space actually used
- Can easily grab next free page to expand page table

■ Disadvantages

- Multi-level page tables are an example of a time-space tradeoff
- Sacrifice address translation time (now 2-level) for space
- Complexity: multi-level schemes are more complex

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.33

EXAMPLE

■ 16KB address space, 64byte pages

■ How large would a one-level page table need to be?

■ 2^{14} (address space) / 2^6 (page size) = 2^8 = 256 (pages)

Flag	Detail
(free)	Address space
(free)	Page size
heap	Virtual address
heap	VPN
(free)	Offset
stack	Page table entry
stack	

A 16-KB Address Space With 64-byte Pages

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.34

EXAMPLE - 2

■ 256 total page table entries (64 bytes each)

■ 1,024 bytes page table size, stored using 64-byte pages = $(1024/64) = 16$ page directory entries (PDEs)

■ Each page directory entry (PDE) can hold 16 page table entries (PTEs) e.g. lookups

■ 16 page directory entries (PDE) x 16 page table entries (PTE) = 256 total PTEs

■ Key Idea: the page table is stored using pages too!

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.35

PAGE DIRECTORY INDEX

■ Now, let's split the page table into two:

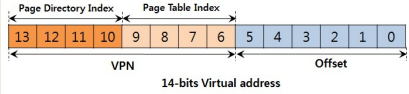
- 8 bit VPN to map 256 pages
- 4 bits for page directory index (PDI – 1st level page table)
- 6 bits offset into 64-byte page

14-bits Virtual address

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.36

PAGE TABLE INDEX

- 4 bits **page directory index** (PDI – 1st level)
- 4 bits **page table index** (PTI – 2nd level)



- To dereference one 64-byte memory page,
 - We need one page directory entry (PDE)
 - One page table Index (PTI) – can address 16 pages

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.37

EXAMPLE - 3

- For this example, how much space is required to store as a single-level page table with any number of PTEs?**
- 16KB address space, 64 byte pages
- 256 page frames, 4 byte page size
- 1,024 bytes required (*single level*)
- How much space is required for a two-level page table with only 4 page table entries (PTEs) ?**
- Page directory = 16 entries x 4 bytes (1 x 64 byte page)
- Page table = 4 entries x 4 bytes (1 x 64 byte page)
- 128 bytes required (2 x 64 byte pages)
 - Savings = using just 12.5% the space !!!

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.38

32-BIT EXAMPLE

- Consider: 32-bit address space, 4KB pages, 2²⁰ pages
- Only 4 mapped pages
- Single level:** 4 MB (we've done this before)
- Two level:** (old VPN was 20 bits, split in half)
 - Page directory = 2¹⁰ entries x 4 bytes = 1 x 4 KB page
 - Page table = 4 entries x 4 bytes (mapped to 1.4KB page)
 - 8KB (8,192 bytes) required
 - Savings = using just .78 % the space !!!
- 100 sparse processes now require < 1MB for page tables

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.39

WE WILL RETURN AT 4:50PM



June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.40

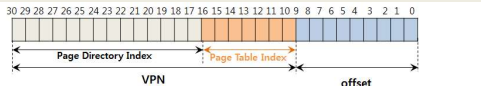
OBJECTIVES – 6/1

- Questions from 5/27
- Assignment 2 – May 31 (*Late Penalty June 2, Closing June 4*)
- Tutorial 2 – Pthread, locks, conditions tutorial – June 4
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Final exam – alternate format
- Quiz 4 – Page Tables
- Chapter 19: Translation Lookaside Buffer (TLB)
 - Hit-to-Miss Ratios
- Chapter 20: Paging: Smaller Tables
 - Smaller Tables, Multi-level Page Tables **N-level Page Tables**
- Chapter 21/22: Beyond Physical Memory
 - Swapping Mechanisms, Swapping Policies

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.41

MORE THAN TWO LEVELS - 2

- Page table entries per page = 512 / 4 = 128
- 7 bytes – 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

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.42

MORE THAN TWO LEVELS - 3

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

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

June 1, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L18.43

MORE THAN TWO LEVELS - 3

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

Can't Store Page Directory with 16K pages, using 512 bytes pages. Pages only dereference 128 addresses (512 bytes / 32 bytes)

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

June 1, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L18.44

MORE THAN TWO LEVELS - 3

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

**Need three level page table:
 Page directory 0 (PD Index 0)
 Page directory 1 (PD Index 1)
 Page Table Index**

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

June 1, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L18.45

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 4 entries on a 512-byte page? (let's say 4 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% !!!

June 1, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L18.46

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 *pmd;
pte_t *pte;
struct page *page;
    
```

June 1, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L18.47

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

June 1, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L18.48

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

June 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L18.49
--------------	---	--------

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 8 status bits, how many bytes are required for each page table entry?

June 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L18.50
--------------	---	--------

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 - stack page
 - 1 - heap 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)?

June 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L18.51
--------------	---	--------

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
 - 8 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

June 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L18.52
--------------	---	--------

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

June 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L18.53
--------------	---	--------

ANSWERS

- #1 - 4096 pages
- #2 - 12 bits
- #3 - 12 bits
- #4 - 4 bytes
- #5 - $4096 \times 4 = 16,384$ 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- $512/16384 = .03125 \rightarrow 3.125\%$


June 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L18.54
--------------	---	--------

OBJECTIVES – 6/1

- Questions from 5/27
- Assignment 2 – May 31 (*Late Penalty June 2, Closing June 4*)
- Tutorial 2 – Pthread, locks, conditions tutorial – June 4
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Final exam – alternate format
- Quiz 4 – Page Tables
- Chapter 19: Translation Lookaside Buffer (TLB)
 - Hit-to-Miss Ratios
- Chapter 20: Paging: Smaller Tables
 - Smaller Tables, Multi-level Page Tables, N-level Page Tables
- **Chapter 21/22: Beyond Physical Memory**
- **Swapping Mechanisms, Swapping Policies**

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.55

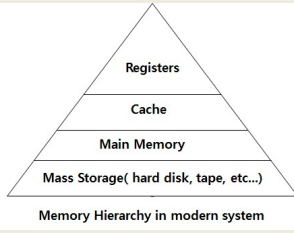
CHAPTER 21/22: BEYOND PHYSICAL MEMORY



June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.56

MEMORY HIERARCHY

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



June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.57

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* . . .

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.58

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>

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.59

OBJECTIVES – 6/1

- Questions from 5/27
- Assignment 2 – May 31 (*Late Penalty June 2, Closing June 4*)
- Tutorial 2 – Pthread, locks, conditions tutorial – June 4
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Final exam – alternate format
- Quiz 4 – Page Tables
- Chapter 19: Translation Lookaside Buffer (TLB)
 - Hit-to-Miss Ratios
- Chapter 20: Paging: Smaller Tables
 - Smaller Tables, Multi-level Page Tables, N-level Page Tables
- **Chapter 21/22: Beyond Physical Memory**
- **Swapping Mechanisms, Swapping Policies**

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.60

SWAP SPACE

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

	PFN 0	PFN 1	PFN 2	PFN 3				
Physical Memory	Proc 0 [VPN 0]	Proc 1 [VPN 2]	Proc 1 [VPN 3]	Proc 2 [VPN 0]				
	Block 0	Block 1	Block 2	Block 3	Block 4	Block 5	Block 6	Block 7
Swap Space	Proc 0 [VPN 1]	Proc 0 [VPN 2]	[Free]	Proc 1 [VPN 0]	Proc 1 [VPN 1]	Proc 3 [VPN 0]	Proc 2 [VPN 1]	Proc 3 [VPN 1]

Physical Memory and Swap Space

June 1, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L18.61

SWAP SPACE - 2

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

```

wllloyd@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

June 1, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L18.62

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 lvsdisplay” command:

```

--- Logical volume ---
LV Path                /dev/ubuntu-vg/swap_1
LV Name                 swap_1
VG Name                 ubuntu-vg
LV UUID                 G10vj6-4H33-2YXY-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

June 1, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L18.63

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

June 1, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L18.64

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
```

June 1, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L18.65

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


June 1, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L18.66

OBJECTIVES – 6/1

- Questions from 5/27
- Assignment 2 – May 31 (*Late Penalty June 2, Closing June 4*)
- Tutorial 2 – Pthread, locks, conditions tutorial – June 4
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Final exam – alternate format
- Quiz 4 – Page Tables
- Chapter 19: Translation Lookaside Buffer (TLB)
 - Hit-to-Miss Ratios
- Chapter 20: Paging: Smaller Tables
 - Smaller Tables, Multi-level Page Tables, N-level Page Tables
- Chapter 21/22: Beyond Physical Memory
 - Swapping Mechanisms **Swapping Policies**

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.67

REPLACEMENT POLICIES



June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.68

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

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.69

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

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.70

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

4 hits

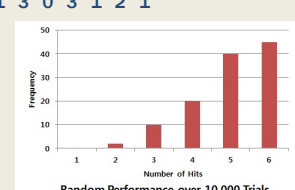
LRU Incorporates history

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.71

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



June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.72

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

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.73

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

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.74

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

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.75

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.

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.76

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)

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.77

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"

June 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.78

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 polliv.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

June 1, 2021 TCS542: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.80

IMPLEMENTING LRU - 2

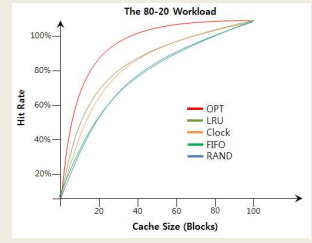

- Harness the Page Table Entry (PTE) Use Bit
 - HW sets to 1 when page is used
 - OS sets to 0
- Clock algorithm (approximate LRU)
 - Refer to pages in a circular list
 - Clock hand points to current page
 - Loops around
 - IF USE_BIT=1 set to USE_BIT = 0
 - IF USE_BIT=0 replace page



June 1, 2021 TCS542: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.81

CLOCK ALGORITHM

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

June 1, 2021 TCS542: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.82

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

June 1, 2021 TCS542: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.83

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

June 1, 2021 TCS542: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L18.84

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

June 1, 2021 TCSS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma L18.85

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

June 1, 2021 TCSS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma L18.86

QUESTIONS

