


TCSS 422: OPERATING SYSTEMS

Multi-Level Page Tables, Beyond Physical Memory



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

December 7, 2021 TCSS422: Operating Systems [Fall 2021]
School of Engineering and Technology, University of Washington Tacoma

1

OBJECTIVES – 12/7

- **Questions from 12/2**
- Memory Segmentation Activity + answers (available in Canvas)
- Assignment 2 – Dec 6 (*Closing Dec 10*)
- Assignment 3 - (Tutorial) Introduction to Linux Kernel Modules
- Final exam – December 14
- Quiz 4 – Page Tables - Due Dec 13
- 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

December 7, 2021 TCSS422: Operating Systems [Fall 2021]
School of Engineering and Technology, University of Washington - Tacoma L17.2

2

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

☰ TCSS 422 A > Assignments

Spring 2021 Search for Assignment

Home

Announcements

Zoom

Syllabus

Assignments

Discussions

▼ Upcoming Assignments

TCSS 422 - Online Daily Feedback Survey - 4/1
Available until Apr 5 at 11:59pm | Due Apr 5 at 10pm | -/1 pts

Quiz 0 - C background survey

December 7, 2021TCSS422: Computer Operating Systems [Fall 2021]
School of Engineering and Technology, University of Washington - TacomaL17.3

3

TCSS 422 - Online Daily Feedback Survey - 4/1

Quiz Instructions

Question 1 0.5 pts

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

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

Question 2 0.5 pts

Please rate the pace of today's class:

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

December 7, 2021TCSS422: Computer Operating Systems [Fall 2021]
School of Engineering and Technology, University of Washington - TacomaL17.4

4

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (27 respondents):
 - 1-mostly review, 5-equal new/review, 10-mostly new
 - **Average - 6.27 (↑ - previous 6.02)**

- Please rate the pace of today's class:
 - 1-slow, 5-just right, 10-fast
 - **Average - 5.58 (↑ - previous 5.46)**

December 7, 2021	TCSS422: Computer Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.5
------------------	--	-------

5

FEEDBACK

- ***I have trouble understanding the math with byte sizes.***
- >>> It is good to review charts and patterns:
 - 8 bits = 1 byte
 - 16 bits = 2 bytes
 - 32 bits = 4 bytes
 - 64 bits = 8 bytes
 - 1,024 bytes = 1 kilobyte (2^{10})
 - 1,024 kilobytes = 1 megabyte (2^{20})
 - 1,024 megabytes = 1 gigabyte (2^{30})
 - 1,024 gigabytes = 1 terabyte (2^{40})
 - 1,024 terrabytes = 1 petabyte (2^{50})

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.6
------------------	---	-------

6

FEEDBACK - 2

- ***For simplicity rounding is often acceptable:***
- 1 kilobyte (2^{10}) = 1,024 bytes → **1,000 bytes**
- 1,024 kilobytes (2^{20}) = 1 megabyte → **1,000,000 bytes**
- 1,024 megabytes = 1 gigabyte (2^{30}) → 1,000,000,000 bytes
- 1,024 gigabytes = 1 terabyte (2^{40}) → 1,000,000,000,000 bytes
- 1,024 terabytes = 1 petabyte (2^{50}) → 1,000,000,000,000,000 bytes

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.7
------------------	---	-------

7

2^1	2	2^{17}	131,072	2^{33}	8,589,934,592	2^{49}	562,949,953,421,312
2^2	4	2^{18}	262,144	2^{34}	17,179,869,184	2^{50}	1,125,899,906,842,624
2^3	8	2^{19}	524,288	2^{35}	34,359,738,368	2^{51}	2,251,799,813,685,248
2^4	16	2^{20} megabyte	1,048,576	2^{36}	68,719,476,736	2^{52}	4,503,599,627,370,496
2^5	32	2^{21}	2,097,152	2^{37}	137,438,953,472	2^{53}	9,007,199,254,740,992
2^6	64	2^{22}	4,194,304	238	274,877,906,944	2^{54}	18,014,398,509,481,984
2^7	128	2^{23}	8,388,608	2^{39}	549,755,813,888	2^{55}	36,028,797,018,963,968
2^8	256	2^{24}	16,777,216	2^{40} terabyte	1,099,511,627,776	2^{56}	72,057,594,037,927,936
2^9	512	2^{25}	33,554,432	2^{41}	2,199,023,255,552	2^{57}	144,115,188,075,855,872
2^{10} kilobyte	1,024	2^{26}	67,108,864	2^{42}	4,398,046,511,104	2^{58}	288,230,376,151,711,744
2^{11}	2,048	2^{27}	134,217,728	2^{43}	8,796,093,022,208	2^{59}	576,460,752,303,423,488
2^{12}	4,096	2^{28}	268,435,456	2^{44}	17,592,186,044,416	2^{60}	1,152,921,504,606,846,976
2^{13}	8,192	2^{29}	536,870,912	2^{45}	35,184,372,088,832	2^{61}	2,305,843,009,213,693,952
2^{14}	16,384	2^{30} gigabyte	1,073,741,824	2^{46}	70,368,744,177,664	2^{62}	4,611,686,018,427,387,904
2^{15}	32,768	2^{31}	2,147,483,648	2^{47}	140,737,488,355,328	2^{63}	9,223,372,036,854,775,808
2^{16}	65,536	2^{32}	4,294,967,296	2^{48}	281,474,976,710,656	2^{64} hubbabyte	18,446,744,073,709,551,616

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.8
------------------	---	-------

8

FEEDBACK - 3

▪ **How many bits are required to index the following amounts of memory?**

1. 1,024 bytes = 1 kilobyte (2^{10})
2. 1,024 kilobytes = 1 megabyte (2^{20})
3. 1,024 megabytes = 1 gigabyte (2^{30})
4. 1,024 gigabytes = 1 terabyte (2^{40})
5. 1,024 terrabytes = 1 petabyte (2^{50})

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.9
------------------	---	-------

9

FEEDBACK - 4

▪ **With paging, we divide an address space in fixed sized pieces (known as the page size)**

▪ **Assuming a computer indexes memory using 1 kilobyte memory pages**

▪ **How many unique pages are required to manage/index memory?**

- 1 kilobyte (2^{10}) of memory
 - 1 page
- 1 megabyte (2^{20}) of memory
 - 1024 pages (2^{10})
- 1 gigabyte (2^{30}) of memory
 - 1,048,576 pages (2^{20})
- 1 terabyte (2^{40}) of memory
 - 1,073,741,824 pages (2^{30})
- 1 petabyte (2^{50}) of memory
 - 1,099,511,627,776 pages (2^{40})

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.10
------------------	---	--------

10

FEEDBACK - 5

- **Assignment 2 Questions:**
- **In chapter 30, the book mentioned the two broken solutions about using a single condition variable. They recommended using two conditional variable locks in order to fix the problem.**
 - Yes with the book there is an empty and fill signal for the bounded buffer when working with multiple producers and multiple consumers
- **But on Thursday, you mentioned that it is not necessary to add more locks (conditions??) but it's up to us to decide.**
- **I wonder why it's not necessary and if there's any big difference.**
 - For each shared bounded buffer in assignment 2, there is only one producer thread and one consumer thread
 - Assignment 2 instead has many unique bounded buffer instances

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.11
------------------	---	--------

11

FEEDBACK - 6

- **Is there more information on "pagemon"? I'm still a little bit confused about what it does and why it is useful.**
 - Pagemon is an interactive tool for browsing memory of a process
- **Are there different ways that we can test/identify for deadlock besides looking at the code?**
 - Try running the program with small bounded buffer sizes
 - Try running the program to generate a large number of primes
 - Try running the program with minimal to no output to the screen
 - Try running the program with no sleep or wait statements
 - Try running the program using as many CPU cores as possible on the virtual machine
 - The fewer the cores, the fewer the synchronization issues
 - 1 core hides deadlock altogether
 - 8 cores is better for testing than 4, than 2, etc.

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.12
------------------	---	--------

12

OBJECTIVES – 12/7

- Questions from 12/2
- **Memory Segmentation Activity + answers (available in Canvas)**
- Assignment 2 – Dec 6 (Closing Dec 10)
- Assignment 3: (Tutorial) Intro to Linux Kernel Modules - Dec 17
- Final exam – Dec 14
- Quiz 4 – Page Tables - Due Dec 13
- 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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.13
------------------	---	--------

13

OBJECTIVES – 12/7

- Questions from 12/2
- Memory Segmentation Activity + answers (available in Canvas)
- **Assignment 2 – Dec 6 (Closing Dec 10)**
- Assignment 3: (Tutorial) Intro to Linux Kernel Modules - Dec 17
- Final exam – Dec 14
- Quiz 4 – Page Tables - Due Dec 13
- 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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.14
------------------	---	--------

14

OBJECTIVES – 12/7

- Questions from 12/2
- Memory Segmentation Activity + answers (available in Canvas)
- Assignment 2 – Dec 6 (Closing Dec 10)
- **Assignment 3: (Tutorial) Intro to Linux Kernel Modules - Dec 17**
- Final exam – Dec 14
- Quiz 4 – Page Tables - Due Dec 13
- 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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.15
------------------	---	--------

15

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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.16
------------------	---	--------

16

OBJECTIVES – 12/7

- Questions from 12/2
- Memory Segmentation Activity + answers (available in Canvas)
- Assignment 2 – Dec 6 (Closing Dec 10)
- Assignment 3: (Tutorial) Intro to Linux Kernel Modules - Dec 17
- **Final exam – Dec 14**
- Quiz 4 – Page Tables - Due Dec 13
- 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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.17
------------------	---	--------

17

FINAL EXAM – DEC 14TH

- Tuesday December 14 from 1:30 to 3:30 pm
 - Final (100 points)
 - SHORT: similar number of questions as the midterm
 - 2-hours
 - Focus on new content - since the midterm (~70% new, 30% before)
- Final Exam Review -
 - Complete Memory Segmentation Activity
 - Complete Quiz 4
 - Practice Final Exam Questions – 2nd hour of Dec 9th class session
 - Individual work
 - 2 pages of notes (any sized paper), double sided
 - Basic calculators allowed
 - NO smartphones, laptop, book, Internet, group work

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.18
------------------	---	--------

18

OBJECTIVES – 12/7

- Questions from 12/2
- Memory Segmentation Activity + answers (available in Canvas)
- Assignment 2 – Dec 6 (Closing Dec 10)
- Assignment 3: (Tutorial) Intro to Linux Kernel Modules - Dec 17
- Final exam – Dec 14
- **Quiz 4 – Page Tables - Due Dec 13**
- 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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.19
------------------	---	--------

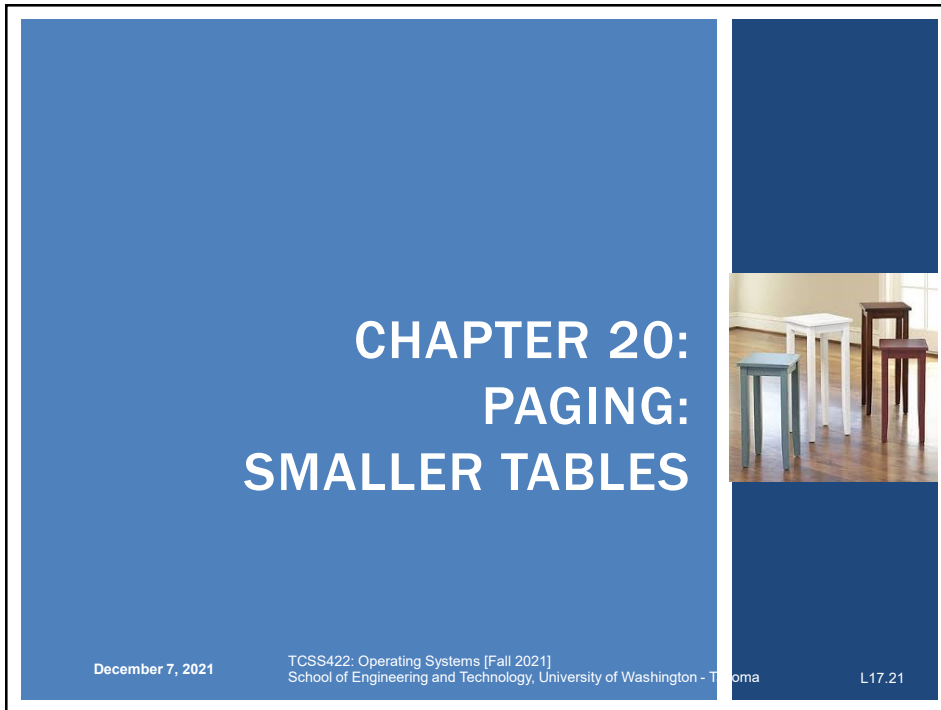
19

OBJECTIVES – 12/7

- Questions from 12/2
- Memory Segmentation Activity + answers (available in Canvas)
- Assignment 2 – Dec 6 (Closing Dec 10)
- Assignment 3: (Tutorial) Intro to Linux Kernel Modules - Dec 17
- Final exam – Dec 14
- Quiz 4 – Page Tables - Due Dec 13
- **Chapter 20: Paging: Smaller Tables**
- Chapter 21/22: Beyond Physical Memory
 - Swapping Mechanisms, Swapping Policies

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.20
------------------	---	--------

20

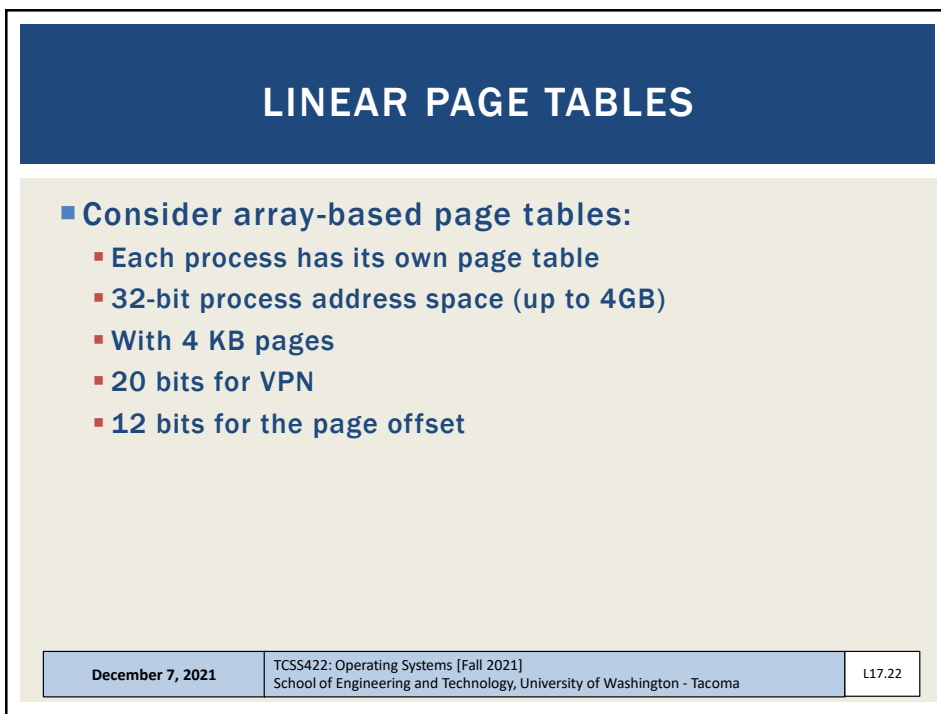


The slide features a large blue background on the left with the title 'CHAPTER 20: PAGING: SMALLER TABLES' in white. On the right, there is a vertical strip with a dark blue top and bottom section, and a central photograph of several small, colorful stools in a room. At the bottom, there is a footer with the date 'December 7, 2021', course information 'TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma', and the slide ID 'L17.21'.

CHAPTER 20: PAGING: SMALLER TABLES

December 7, 2021 TCSS422: Operating Systems [Fall 2021]
School of Engineering and Technology, University of Washington - Tacoma L17.21

21



The slide has a dark blue header with the title 'LINEAR PAGE TABLES'. Below the header is a light beige background containing a bulleted list of points. At the bottom, there is a footer with the date 'December 7, 2021', course information 'TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma', and the slide ID 'L17.22'.

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

December 7, 2021 TCSS422: Operating Systems [Fall 2021]
School of Engineering and Technology, University of Washington - Tacoma L17.22

22

LINEAR PAGE TABLES - 2

- Page tables stored in RAM
- Support potential storage of 2^{20} 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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.23
------------------	---	--------

23

LINEAR PAGE TABLES - 2

- Page tables stored in RAM
- Support potential storage of 2^{20} 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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.24
------------------	---	--------

24

OBJECTIVES – 12/7

- Questions from 12/2
- Memory Segmentation Activity + answers (available in Canvas)
- Assignment 2 – Dec 6 (Closing Dec 10)
- Assignment 3: (Tutorial) Intro to Linux Kernel Modules - Dec 17
- Final exam – Dec 14
- Quiz 4 – Page Tables - Due Dec 13
- 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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.25
------------------	---	--------

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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.26
------------------	---	--------

26

PAGE TABLES: WASTED SPACE

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

Page Table
Virtual Address Space

code
0
1
2
3
heap
4
5
6
7
8
9
10
11
12
stack
13
14

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

December 7, 2021
TCSS422: Operating Systems [Fall 2021]
School of Engineering and Technology, University of Washington - Tacoma
L17.27

27

PAGE TABLES: WASTED SPACE

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

Page Table
Virtual Address Space

code
0
1
2
heap
4
5
6
7
8
9
10
11
12
stack
13
14

A 16KB Address Space with 1KB Pages

PFN	valid	prot	present	dirty
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

Most of the page table is unused and full of wasted space. (73%)

December 7, 2021
TCSS422: Operating Systems [Fall 2021]
School of Engineering and Technology, University of Washington - Tacoma
L17.28

28

OBJECTIVES – 12/7

- Questions from 12/2
- Memory Segmentation Activity + answers (available in Canvas)
- Assignment 2 – Dec 6 (Closing Dec 10)
- Assignment 3: (Tutorial) Intro to Linux Kernel Modules - Dec 17
- Final exam – Dec 14
- Quiz 4 – Page Tables - Due Dec 13
- 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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.29
------------------	---	--------

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!**

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.30
------------------	---	--------

30

MULTI-LEVEL PAGE TABLES - 2

■ Add level of indirection, the “page directory”

Linear Page Table

PBTR 201

valid	prot	PFN
1	rx	12
1	rx	13
0	-	-
1	rw	100
0	-	-
0	-	-
0	-	-
0	-	-
0	-	-
1	rw	86
1	rw	15

PFN201

Multi-level Page Table

PBTR 200

valid	PFN
1	201
0	-
0	-
1	203

The Page Directory

PFN200

valid	prot	PFN
1	rx	12
1	rx	13
0	-	-
1	rw	100

PFN201

[Page 1 of PT: Not Allocated]

valid	prot	PFN
0	-	-
0	-	-
1	rw	86
1	rw	15

PFN204

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

December 7, 2021
TCSS422: Operating Systems [Fall 2021]
 School of Engineering and Technology, University of Washington - Tacoma
L17.31

31

MULTI-LEVEL PAGE TABLES - 2

■ Add level of indirection, the “page directory”

Linear Page Table

PBTR 201

valid	prot	PFN
0	-	-
0	-	-
1	rw	86
1	rw	15

PFN203

Multi-level Page Table

PBTR 200

valid	PFN
0	-
0	-
1	203

The Page Directory

PFN200

valid	prot	PFN
0	-	-
0	-	-
1	rw	86
1	rw	15

PFN204

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

December 7, 2021
TCSS422: Operating Systems [Fall 2021]
 School of Engineering and Technology, University of Washington - Tacoma
L17.32

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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.33
------------------	---	--------

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)

0000 0000	code
0000 0001	code
-	(free)
	(free)
	heap
	heap
	(free)
	(free)
	stack
1111 1111	stack

Flag	Detail
Address space	16 KB
Page size	64 byte
Virtual address	14 bit
VPN	8 bit
Offset	6 bit
Page table entry	2^8 (256)

A 16-KB Address Space With 64-byte Pages

13	12	11	10	9	8	7	6	5	4	3	2	1	0
←----->											←----->		
											Offset		

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.34
------------------	---	--------

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!**

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.35
------------------	---	--------

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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.36
------------------	---	--------

36

PAGE TABLE INDEX

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

14-bits Virtual address

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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.37
------------------	---	--------

37

Multi-level page table sketch

38

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 = 16 entries (4 used) x 4 bytes (1 x 64 byte page)
 - 128 bytes required (2 x 64 byte pages)
 - Savings = using just 12.5% the space !!!

December 7, 2021

TCSS422: Operating Systems [Fall 2021]
School of Engineering and Technology, University of Washington - Tacoma

L17.39

39

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

Storage requirement: bytes required (*single level*)

40

How much space is required for a **two-level** page table with only 4 page table entries (PTEs) ? (one page each for code segment, stack segment, heap segment, data segment)
16KB address space, 64 byte pages, 256 page frames, 4 byte page size

Page directory = 16 entries x 4 bytes (1 x 64 byte page)
Page table = 16 entries (4 used) x 4 bytes (1 x 64 byte page)
Store requirement = 128 bytes required (2 x 64 byte pages)
Savings =

41

32-BIT EXAMPLE

- Consider: 32-bit address space, 4KB pages, 2^{20} 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 = 2^{10} 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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.42
------------------	---	--------

42

OBJECTIVES – 12/7

- Questions from 12/2
- Memory Segmentation Activity + answers (available in Canvas)
- Assignment 2 – Dec 6 (Closing Dec 10)
- Assignment 3: (Tutorial) Intro to Linux Kernel Modules - Dec 17
- Final exam – Dec 14
- Quiz 4 – Page Tables - Due Dec 13
- 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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.43
------------------	---	--------

43

MORE THAN TWO LEVELS - 2

- Page table entries per page = $512 / 4 = 128$
- 7 bytes – for page table index (PTI)

← Page Directory Index
← Page Table Index
← offset

VPN
offset

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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.44
------------------	---	--------

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

Flag	Detail
Virtual address	30 bit
Page size	512 byte
VPN	21 bit
Offset	9 bit
Page entry per page	128 PTEs

→ $\log_2 128 = 7$

December 7, 2021
TCSS422: Operating Systems [Fall 2021]
 School of Engineering and Technology, University of Washington - Tacoma
L17.45

45

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

→ $\log_2 128 = 7$

December 7, 2021
TCSS422: Operating Systems [Fall 2021]
 School of Engineering and Technology, University of Washington - Tacoma
L17.46

46

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	→ $\log_2 128 = 7$

December 7, 2021
TCSS422: Operating Systems [Fall 2021]
 School of Engineering and Technology, University of Washington - Tacoma
L17.47

47

MORE THAN TWO LEVELS - 4

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

30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

← PD Index 0 ← PD Index 1 ← Page Table Index

VPN

- 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% !!!

December 7, 2021
TCSS422: Operating Systems [Fall 2021]
 School of Engineering and Technology, University of Washington - Tacoma
L17.48

48

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;

```

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.49
------------------	---	--------

49

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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.50
------------------	---	--------

50


**WE WILL RETURN AT
2:40PM**



December 7, 2021 TCSS422: Operating Systems [Fall 2021]
School of Engineering and Technology, University of Washington - Tacoma L17.51

51

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

December 7, 2021 TCSS422: Operating Systems [Fall 2021]
School of Engineering and Technology, University of Washington - Tacoma L17.52

52

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?

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.53
------------------	---	--------

53

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)?

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.54
------------------	---	--------

54

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

December 7, 2021

TCSS422: Operating Systems [Fall 2021]
School of Engineering and Technology, University of Washington - Tacoma

L17.55

55

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

December 7, 2021

TCSS422: Operating Systems [Fall 2021]
School of Engineering and Technology, University of Washington - Tacoma

L17.56

56

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- $512/16384 = .03125 \rightarrow 3.125\%$

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.57
------------------	---	--------

57


OBJECTIVES - 12/7

- Questions from 12/2
- Memory Segmentation Activity + answers (available in Canvas)
- Assignment 2 - Dec 6 (Closing Dec 10)
- Assignment 3: (Tutorial) Intro to Linux Kernel Modules - Dec 17
- Final exam - Dec 14
- Quiz 4 - Page Tables - Due Dec 13
- 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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.58
------------------	---	--------

58

CHAPTER 21/22: BEYOND PHYSICAL MEMORY

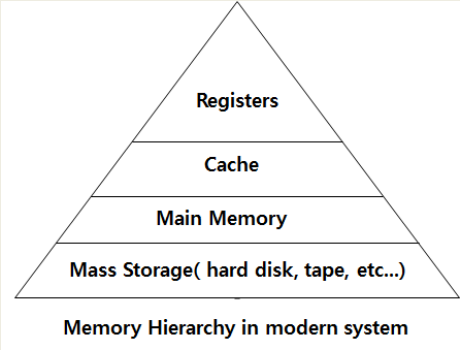


December 7, 2021 TCSS422: Operating Systems [Fall 2021]
School of Engineering and Technology, University of Washington - Tacoma L17.59

59

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

December 7, 2021 TCSS422: Operating Systems [Fall 2021]
School of Engineering and Technology, University of Washington - Tacoma L17.60

60

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. . .*

December 7, 2021
TCSS422: Operating Systems [Fall 2021]
 School of Engineering and Technology, University of Washington - Tacoma
L17.61

61

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>

December 7, 2021
TCSS422: Operating Systems [Fall 2021]
 School of Engineering and Technology, University of Washington - Tacoma
L17.62

62

OBJECTIVES – 12/7

- Questions from 12/2
- Memory Segmentation Activity + answers (available in Canvas)
- Assignment 2 – Dec 6 (Closing Dec 10)
- Assignment 3: (Tutorial) Intro to Linux Kernel Modules - Dec 17
- Final exam – Dec 14
- Quiz 4 – Page Tables - Due Dec 13
- 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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.63
------------------	---	--------

63

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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.64
------------------	---	--------

64

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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.65
------------------	---	--------

65

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 lvs” command:

```
--- Logical volume ---
LV Path                /dev/ubuntu-vg/swap_1
LV Name                 swap_1
VG Name                 ubuntu-vg
LV UUID                 G1ovj6-4M33-2YXY-VETH-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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.66
------------------	---	--------

66

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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.67
------------------	---	--------

67

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
```

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.68
------------------	---	--------

68

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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.69
------------------	---	--------

69

OBJECTIVES – 12/7

- Questions from 12/2
- Memory Segmentation Activity + answers (available in Canvas)
- Assignment 2 – Dec 6 (Closing Dec 10)
- Assignment 3: (Tutorial) Intro to Linux Kernel Modules - Dec 17
- Final exam – Dec 14
- Quiz 4 – Page Tables - Due Dec 13
- 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**


December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.70
------------------	---	--------

70

REPLACEMENT POLICIES

December 7, 2021

TCSS422: Operating Systems [Fall 2021]
 School of Engineering and Technology, University of Washington - Tacoma



L17.7
1

71

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

December 7, 2021

TCSS422: Operating Systems [Fall 2021]
 School of Engineering and Technology, University of Washington - Tacoma

L17.72

72

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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.73
------------------	---	--------

73

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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.74
------------------	---	--------

74

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

Number of Hits	Frequency
1	0
2	2
3	10
4	20
5	40
6	45

December 7, 2021

TCSS422: Operating Systems [Fall 2021]
 School of Engineering and Technology, University of Washington - Tacoma

L17.75

75

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

December 7, 2021

TCSS422: Operating Systems [Fall 2021]
 School of Engineering and Technology, University of Washington - Tacoma

L17.76

76

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

December 7, 2021 TCSS422: Operating Systems [Fall 2021] L17.7

77

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

December 7, 2021 TCSS422: Operating Systems [Fall 2021] L17.8

78

WORKLOAD EXAMPLES: NO-LOCALITY

- **No-Locality (Random Access) Workload**
 - Perform 10,000 random page accesses
 - Across set of 100 memory pages

The graph shows Hit Rate on the y-axis (0% to 100%) and Cache Size (Blocks) on the x-axis (0 to 100). Four lines represent different replacement policies: OPT (red), LRU (blue), FIFO (orange), and RAND (green). All lines start at (0,0) and end at (100,100). The RAND line is a straight diagonal line. The other three lines (OPT, LRU, FIFO) are curves that rise more steeply initially and then level off, all reaching 100% hit rate at 100 blocks.

When the cache is large enough to fit the entire workload, it doesn't matter which policy you use.

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.79
------------------	---	--------

79

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)

The graph shows Hit Rate on the y-axis (0% to 100%) and Cache Size (Blocks) on the x-axis (0 to 100). Four lines represent different replacement policies: OPT (red), LRU (blue), FIFO (orange), and RAND (green). All lines start at (0,0) and end at (100,100). The RAND line is a straight diagonal line. The other three lines (OPT, LRU, FIFO) are curves that rise more steeply initially and then level off, all reaching 100% hit rate at 100 blocks. LRU (blue) is the highest curve, followed by OPT (red), then FIFO (orange), and RAND (green) is the lowest curve.

LRU is more likely to hold onto hot pages
(recalls history)

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.80
------------------	---	--------

80

WORKLOAD EXAMPLES: SEQUENTIAL

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

Cache Size (Blocks)	OPT Hit Rate (%)	LRU Hit Rate (%)	FIFO Hit Rate (%)	RAND Hit Rate (%)
0	0	0	0	0
20	20	0	10	10
40	40	0	30	30
45	45	0	100	100
50	100	0	100	100
100	100	100	100	100

Random performs better than FIFO and LRU for cache sizes < 50

Algorithms should provide "scan resistance"

December 7, 2021TCSS422: Operating Systems [Fall 2021]
School of Engineering and Technology, University of Washington - TacomaL17.81

81

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

82

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


December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.83
------------------	---	--------

83

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



December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.84
------------------	---	--------

84

CLOCK ALGORITHM

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

Cache Size (Blocks)	OPT	LRU	Clock	FIFO	RAND
0	0%	0%	0%	0%	0%
20	85%	75%	70%	65%	60%
40	95%	85%	80%	75%	70%
60	98%	90%	85%	80%	75%
80	99%	92%	88%	82%	78%
100	100%	95%	90%	85%	80%

December 7, 2021TCSS422: Operating Systems [Fall 2021]
School of Engineering and Technology, University of Washington - TacomaL17.85

85

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

December 7, 2021TCSS422: Operating Systems [Fall 2021]
School of Engineering and Technology, University of Washington - TacomaL17.86

86

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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.87
------------------	---	--------

87

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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.88
------------------	---	--------

88


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

December 7, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.89
------------------	---	--------

89

QUESTIONS



90