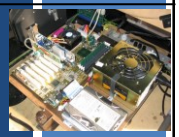
TCSS 422: OPERATING SYSTEMS



Memory Virtualization IV:
Translation Lookaside Buffer (TLB),
Smaller Tables,
Multi-Level Page Tables,
Beyond Physical Memory

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

1

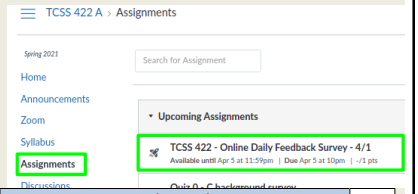
OBJECTIVES – 3/10

- **Questions from 3/5**
- Memory Segmentation Activity + answers (available in Canvas)
- Assignment 2 - March 12 AOE
- Assignment 3 (as a Tutorial) - March 20 AOE
- Final exam – Thursday March 19 @ 3:40pm
- Quiz 4 – Page Tables - Due March 12 AOE
- Chapter 19: Translation Lookaside Buffer (TLB)
  - TLB Algorithm, 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

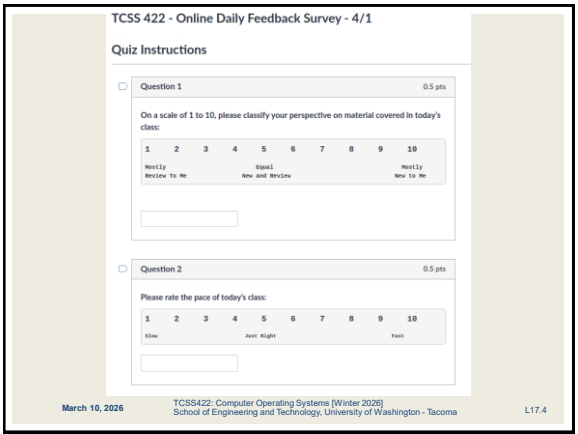
2

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



3



4

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (42 of 46 respondents (3 online) – 91.3%):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 4.73 (↓ - previous 6.87)**

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 4.93 (↓ - previous 5.55)**

5

FEEDBACK FROM 3/5

- ***Does internal memory fragmentation occur while using pages (paging) ?***
  - Yes, if the page(s) are sparsely used
  - Consider helloworld.c
  - 4 KB – code segment page (hello world code is very small)
  - 4 KB – stack segment page (hello world doesn't use functions)
  - 4 KB – heap segment page (hello world doesn't use heap)
  - 4 KB – data segment page (hello world has no global data)
  - Memory allocated = 16 KB
  - Memory required = less than 1 KB
- ***The review was nice. Nothing really stands out to me right now.***

7

REVIEW - 2

- ***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 (2^10)***
- ***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)

8

OBJECTIVES – 3/10

- Questions from 3/5
- **Memory Segmentation Activity + answers (available in Canvas)**
- Assignment 2 - March 12 AOE
- A3: (Tutorial) Intro to Linux Kernel Modules - June 10 AOE
- Final exam – Thursday March 19 @ 3:40pm
- Quiz 4 – Page Tables - Due March 12 AOE
- Chapter 19: Translation Lookaside Buffer (TLB)
  - TLB Algorithm, 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

9

OBJECTIVES – 3/10

- Questions from 3/5
- Memory Segmentation Activity + answers (available in Canvas)
- **Assignment 2 - March 12 AOE**
- A3: (Tutorial) Intro to Linux Kernel Modules - June 10 AOE
- Final exam – Thursday March 19 @ 3:40pm
- Quiz 4 – Page Tables - Due March 12 AOE
- Chapter 19: Translation Lookaside Buffer (TLB)
  - TLB Algorithm, 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

10

OBJECTIVES – 3/10

- Questions from 3/5
- Memory Segmentation Activity + answers (available in Canvas)
- Assignment 2 - March 12 AOE
- **A3: (Tutorial) Intro to Linux Kernel Modules - June 10 AOE**
- Final exam – Thursday March 19 @ 3:40pm
- Quiz 4 – Page Tables - Due March 12 AOE
- Chapter 19: Translation Lookaside Buffer (TLB)
  - TLB Algorithm, 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

11

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

12

OBJECTIVES – 3/10

- Questions from 3/5
- Memory Segmentation Activity + answers (available in Canvas)
- Assignment 2 - March 12 AOE
- A3: (Tutorial) Intro to Linux Kernel Modules - June 10 AOE
- **Final exam – Thursday March 19 @ 3:40pm**
- Quiz 4 – Page Tables - Due March 12 AOE
- Chapter 19: Translation Lookaside Buffer (TLB)
  - TLB Algorithm, 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

13

FINAL EXAM – THURSDAY MARCH 19 @ 3:40PM[TH]

- Thursday June 12 from 3:40 to 5:40 pm
  - Final (100 points)
  - Similar number of questions as the midterm
  - 2-hours
  - Focus on new content - since the midterm (~70% new, 30% before)
- Final Exam Review -
  - Complete Memory Segmentation Activity
  - Complete Quiz 4
  - Practice Final Exam Questions – 2nd hour of June 5th class session
  - Individual work
  - 3 pages of notes (any sized paper), double sided
  - Basic calculators allowed
  - NO smartphones, laptop, book, Internet, group work

14

OBJECTIVES – 3/10

- Questions from 3/5
- Memory Segmentation Activity + answers (available in Canvas)
- Assignment 2 - March 12 AOE
- A3: (Tutorial) Intro to Linux Kernel Modules - June 10 AOE
- Final exam – Thursday March 19 @ 3:40pm
- **Quiz 4 – Page Tables - Due March 12 AOE**
- Chapter 19: Translation Lookaside Buffer (TLB)
  - TLB Algorithm, 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

15

TUTORIAL 3 - FILE SYSTEMS

- **(Optional, Extra Credit)**
- Due Saturday March 21 AOE time
- In Extra Credit Category
- Earn up to 2% extra credit added to overall course credit
- Topics:
  - Exploring the File API (Chapter 39)
  - File System Symbolic Links in Linux
  - File System types: ext2, ext4
  - Testing Performance Impact of File System Journaling w/ Sysbench
  - iNodes, iNode density on a file system

16

CATCH UP FROM LECTURE 15/16

- Switch to Lecture 15 Slides
- Slides L15.33 to L15.81
  (Chapter 19 – Translation Lookaside Buffer - TLB)
  (Chapter 20 – Paging – Smaller Tables)

17

WE WILL RETURN AT 5:00 PM

18

OBJECTIVES – 3/10

- Questions from 3/5
- Memory Segmentation Activity + answers (available in Canvas)
- Assignment 2 - March 12 AOE
- A3: (Tutorial) Intro to Linux Kernel Modules - June 10 AOE
- Final exam – Thursday March 19 @ 3:40pm
- Quiz 4 – Page Tables - Due March 12 AOE
- Chapter 19: Translation Lookaside Buffer (TLB)
  - TLB Algorithm, 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

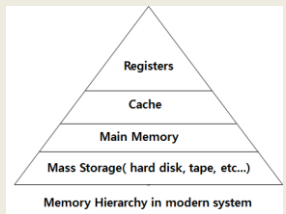
19

CHAPTER 21/22: BEYOND PHYSICAL MEMORY

20

MEMORY HIERARCHY

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



Memory Hierarchy in modern system

21

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. . .*

22

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

23

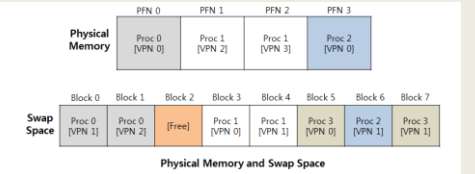
OBJECTIVES – 3/10

- Questions from 3/5
- Memory Segmentation Activity + answers (available in Canvas)
- Assignment 2 - March 12 AOE
- A3: (Tutorial) Intro to Linux Kernel Modules - June 10 AOE
- Final exam – Thursday March 19 @ 3:40pm
- Quiz 4 – Page Tables - Due March 12 AOE
- Chapter 19: Translation Lookaside Buffer (TLB)
  - TLB Algorithm, 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

24

SWAP SPACE

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



Physical Memory and Swap Space

25

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

26

SWAP SPACE - 3

- Swap space lives on a separate logical volume in Ubuntu Linux that is managed separately from the root file system
- Check logical volumes with "sudo lvdisplay" command:

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

- See also "lvm lvs" command

27

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

28

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
```


29

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

30

OBJECTIVES – 3/10

- Questions from 3/5
- Memory Segmentation Activity + answers (available in Canvas)
- Assignment 2 - March 12 AOE
- A3: (Tutorial) Intro to Linux Kernel Modules - June 10 AOE
- Final exam – Thursday March 19 @ 3:40pm
- Quiz 4 – Page Tables - Due March 12 AOE
- Chapter 19: Translation Lookaside Buffer (TLB)
  - TLB Algorithm, 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**

31

REPLACEMENT POLICIES

32

CACHE MANAGEMENT

- Replacement policies apply to "any" cache
- Goal is to minimize the number of misses
- Average memory access time (AMAT) can be estimated:

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

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

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

33

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

34

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

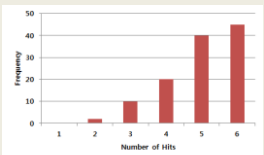
- What is the hit/miss ratio? 4 hits
- How is FIFO different than LRU? LRU incorporates history

35

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



36

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

37

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

38

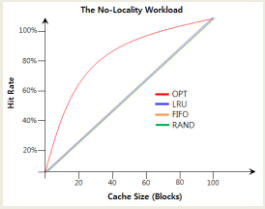
Consider a 3-element cache. With an LRU replacement policy, how many hits occur with the following page access sequence:
1 2 0 1 3 1 2 0 2 1 3

2 hits
3 hits
4 hits
5 hits
6 hits

39

WORKLOAD EXAMPLES: NO-LOCALITY

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

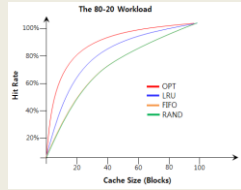


When the cache is large enough to fit the entire workload, it doesn't matter which policy you use.

40

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)



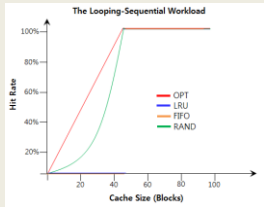
LRU is more likely to hold onto hot pages

(recalls history)

41

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"

42

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

43

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

44

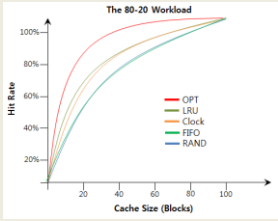
IMPLEMENTING LRU - 2

- 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

45

CLOCK ALGORITHM

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



46

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

47

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

48

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

49

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

50

QUESTIONS

51