TCSS 422: OPERATING SYSTEMS

Beyond Physical Memory, Review for Final Exam

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

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

1

2022 - TCSS 498/499 (ANY QUARTER) UNDERGRADUATE READING/RESEARCH IN CSS

- Independent study in “cloud computing”
- Work to collaboratively draft a proposal and submit to Dr. Nascimento, CSS Chair for approval
- Focus on variety of topics related to cloud/distributed systems
- Variable credits from 1 to 5
- Involves participation in weekly research group meeting
  - Winter 2022: Wednesday at 12:30p
- Usually 1 or 2 one-on-one or small group meeting during week
- Contact by email if interested
- Identify preferred quarter(s)
- Number of credits
- Can take up to 10 credits TCSS 498/499 - CSS elective credits

December 9, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L18.2

2

COURSE EVALUATION: TCSS 422 A FALL 2021

- Please complete the course evaluation survey at:
- TCSS 422 A - Computer Operating Systems:
- https://uwt.iasystem.org/survey/109423
- **Special features this quarter in TCSS 422:**
- Class sessions LIVE streamed over Zoom, with all lecture recordings made available shortly after class (use of 2 computers)
- No mandatory graded in class activities this quarter to maximize attendance/participation flexibility for covid and commuting – *(enables mostly asynchronous participation)*
- OBS Studio software used to provide different "scenes" that integrate different screen captures with camera, chat, and displays
- Slide refinements to improve online delivery
- Assignment 3 graded as a Quiz/Tutorial: Kernel Module programming
- Extra credit for paperless daily feedback surveys
- Tutorial 3 on File Systems optional for extra credit
- Assignment 2– new assignment, single producer, single consumer, multiple buffer provides revised scope for covid-19, with primary focus on pthreads, locking, and bounded buffer

3

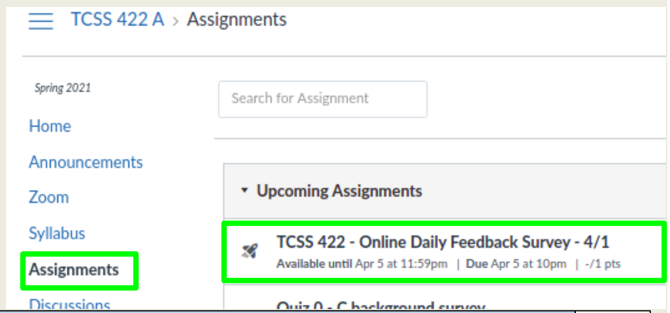
OBJECTIVES – 12/9

- **Questions from 12/7**
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Memory Segmentation Activity + answers (available in Canvas)
- Quiz 4 – Page Tables
- Final exam – December 14
- Tutorial 3 - File Systems (Optional, Extra Credit)
- Chapter 21/22: Beyond Physical Memory
  - Swapping Mechanisms, Swapping Policies
- Practice Questions for Final Exam

4

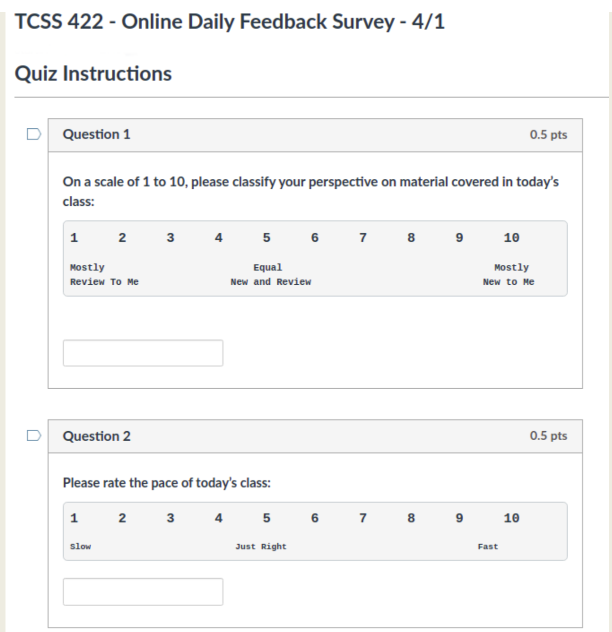
ONLINE DAILY FEEDBACK SURVEY

- 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



December 9, 2021 | TCSS422: Computer Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma | L18.5

5



December 9, 2021 | TCSS422: Computer Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma | L18.6

6

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (24 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 5.80 (↓ - previous 6.27)**

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.64 (↑ - previous 5.58)**

7

FEEDBACK

- ***What is the update on assignment grading?***
- Recently graded: Active Reading Quiz Chapter 9
- Current: Tutorial 1 and Quiz 3
- Next: Assignment 2
- Remainder is online quizzes and the Final Exam

8

OBJECTIVES – 12/9

- Questions from 12/7
- **Assignment 3: (Tutorial) Introduction to Linux Kernel Modules**
- Memory Segmentation Activity + answers (available in Canvas)
- Quiz 4 – Page Tables
- Final exam – December 14
- Tutorial 3 - File Systems (Optional, Extra Credit)
- Chapter 21/22: Beyond Physical Memory
  - Swapping Mechanisms, Swapping Policies
- Practice Questions for Final Exam

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

OBJECTIVES – 12/9

- Questions from 12/7
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- **Memory Segmentation Activity + answers (available in Canvas)**
- Quiz 4 – Page Tables
- Final exam – December 14
- Tutorial 3 - File Systems (Optional, Extra Credit)
- Chapter 21/22: Beyond Physical Memory
  - Swapping Mechanisms, Swapping Policies
- Practice Questions for Final Exam

OBJECTIVES – 12/9

- Questions from 12/7
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Memory Segmentation Activity + answers (available in Canvas)
- **Quiz 4 – Page Tables**
- Final exam – December 14
- Tutorial 3 - File Systems (Optional, Extra Credit)
- Chapter 21/22: Beyond Physical Memory
  - Swapping Mechanisms, Swapping Policies
- Practice Questions for Final Exam

OBJECTIVES – 12/9

- Questions from 12/7
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Memory Segmentation Activity + answers (available in Canvas)
- Quiz 4 – Page Tables
- **Final exam – December 14**
- Tutorial 3 - File Systems (Optional, Extra Credit)
- Chapter 21/22: Beyond Physical Memory
  - Swapping Mechanisms, Swapping Policies
- Practice Questions for Final Exam

FINAL EXAM – DEC 14^{TH}

- 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 – 2^{nd} hour of Dec 9^{th} class session
  - Individual work
  - 2 pages of notes (any sized paper), double sided
  - Basic calculators allowed
  - NO smartphones, laptop, book, Internet, group wowkr

OBJECTIVES – 12/9

- Questions from 12/7
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Memory Segmentation Activity + answers (available in Canvas)
- Quiz 4 – Page Tables
- Final exam – December 14
- **Tutorial 3 - File Systems (Optional, Extra Credit)**
- Chapter 21/22: Beyond Physical Memory
  - Swapping Mechanisms, Swapping Policies
- Practice Questions for Final Exam

OBJECTIVES – 12/9

- Questions from 12/7
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Memory Segmentation Activity + answers (available in Canvas)
- Quiz 4 – Page Tables
- Final exam – December 14
- Tutorial 3 - File Systems (Optional, Extra Credit)
- **Chapter 21/22: Beyond Physical Memory**
  - Swapping Mechanisms, Swapping Policies
- Practice Questions for Final Exam

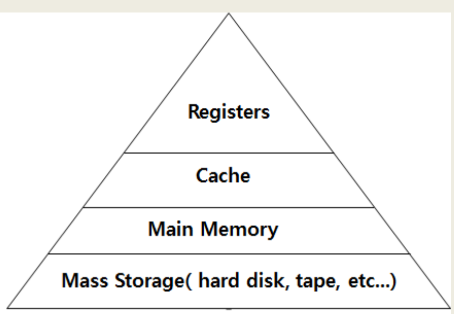
CHAPTER 21/22: BEYOND PHYSICAL MEMORY

December 9, 2021 | TCSS422: Operating Systems [Fall 2021]
School of Engineering and Technology, University of Washington - Tacoma | L18.17

17

MEMORY HIERARCHY

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



Memory Hierarchy in modern system

December 9, 2021 | TCSS422: Operating Systems [Fall 2021]
School of Engineering and Technology, University of Washington - Tacoma | L18.18

18

MOTIVATION FOR EXPANDING THE ADDRESS SPACE

- Provide the illusion of an address space larger than physical RAM
- For a single process
  - Convenience
  - Ease of use
- For multiple processes
  - Large virtual memory space supports running *many concurrent processes. . .*

LATENCY TIMES

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

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

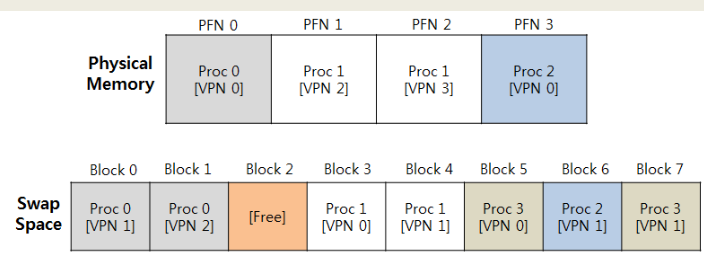
- *Latency numbers every programmer should know*
- From: https://gist.github.com/jboner/2841832#file-latency-txt

OBJECTIVES – 12/9

- Questions from 12/7
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Memory Segmentation Activity + answers (available in Canvas)
- Quiz 4 – Page Tables
- Final exam – December 14
- Tutorial 3 - File Systems (Optional, Extra Credit)
- Chapter 21/22: Beyond Physical Memory
  - **Swapping Mechanisms** Swapping Policies
- Practice Questions for Final Exam

SWAP SPACE

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



Physical Memory and Swap Space

SWAP SPACE - 2

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

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

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

SWAP SPACE - 3

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

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

- See also “lvm lvs” command

PAGE LOCATION

- Memory pages are:
  - Stored in memory
  - Swapped to disk
- Present bit
  - In the page table entry (PTE) indicates if page is present
- Page fault
  - Memory page is accessed, but has been swapped to disk

25

PAGE FAULT

- OS steps in to handle the page fault
- Loading page from disk requires a free memory page
- Page-Fault Algorithm

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


26

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

27

OBJECTIVES – 12/9

- Questions from 12/7
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Memory Segmentation Activity + answers (available in Canvas)
- Quiz 4 – Page Tables
- Final exam – December 14
- Tutorial 3 - File Systems (Optional, Extra Credit)
- Chapter 21/22: Beyond Physical Memory
  - Swapping Mechanisms, **Swapping Policies**
- Practice Questions for Final Exam

28

REPLACEMENT POLICIES

29

CACHE MANAGEMENT

- Replacement policies apply to "any" cache
- Goal is to minimize the number of misses
- **A**verage **m**emory **a**ccess **t**ime **(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

30

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

31

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?
- How is FIFO different than LRU?

4 hits

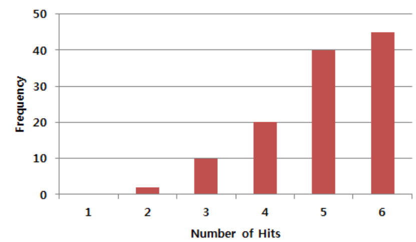
LRU incorporates history

32

RANDOM REPLACEMENT

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

0 1 2 0 1 3 0 3 1 2 1



Random Performance over 10,000 Trials

33

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

34

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

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

December 9, 2021 | TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma | L18.35

35

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

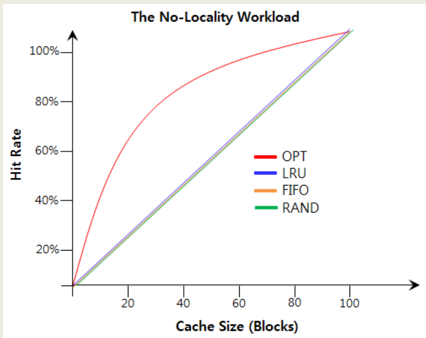
Start the presentation to see live content. For screen share software, share the entire screen. Get help at pollev.com/app

December 9, 2021 | TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma | L18.36

36

WORKLOAD EXAMPLES: NO-LOCALITY

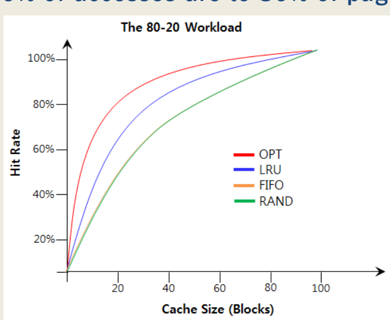
- No-Locality (Random Access) Workload
  - Perform 10,000 random page accesses
  - Across set of 100 memory pages



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

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)

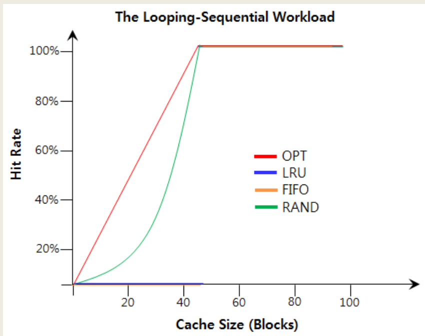


LRU is more likely to hold onto hot pages

(recalls history)

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"

December 9, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L18.39

39

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**

40

WE WILL RETURN AT 2:40PM

December 9, 2021 | TCSS422: Operating Systems [Fall 2021]
School of Engineering and Technology, University of Washington - Tacoma | L18.41

41

OBJECTIVES – 12/9

- Questions from 12/7
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Memory Segmentation Activity + answers (available in Canvas)
- Quiz 4 – Page Tables
- Final exam – December 14
- Tutorial 3 - File Systems (Optional, Extra Credit)
- Chapter 21/22: Beyond Physical Memory
  - Swapping Mechanisms, Swapping Policies
- **Practice Questions for Final Exam**

December 9, 2021 | TCSS422: Operating Systems [Fall 2021]
School of Engineering and Technology, University of Washington - Tacoma | L18.42

42

QUESTION 1 – BASE AND BOUNDS

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

Scenario 1

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

43

Q1 - 2

Scenario 2

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

Scenario 3

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

44

QUESTION 2 – SINGLE-LEVEL PAGE TABLE

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

Q2 - 2

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

QUESTION 3 - TWO-LEVEL PAGE TABLE

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

Q3 - 2

- (d) Assuming each page table entry (PTE) requires 4 bytes of memory, how many extra bits are available for status bits?
- (e) HelloWorld.c consists of 4 memory pages. One code page, one heap page, one data segment page, and one stack segment page. How large is the two-level page table in bytes with the structure described above that could index the all 4 memory pages of HelloWorld.c?
  Hint: There should be 2 tables, a page directory, and a page table.
- *(f) Assuming the same page table as for HelloWorld.c, using the exact same two-level page table, how large in bytes could the program grow to before needing to expand the page table?*

QUESTION 4 – CACHE TRACING

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

A. Optimal policy

Arrival sequence:

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

Working Cache
Cache 1:
Cache 2:
Cache 3:

Hits: ______________

Misses: ______________

Q4 - 2

B. FIFO policy

Arrival sequence:

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

Working Cache
Cache 1:
Cache 2:
Cache 3:

Hits: ______________

Misses: ______________

C. LRU policy

Arrival sequence:

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

Working Cache
Cache 1:
Cache 2:
Cache 3:

Hits: ______________

Misses: ______________

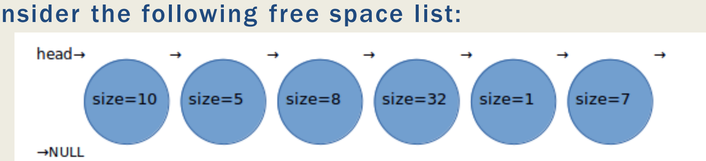
QUESTION 5 – FREE SPACE MANAGEMENT

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

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


Q5 - 2

- Consider the following free space list:



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

Q5 - 3

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

December 9, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L18.53

53

QUESTIONS

54

QUESTIONS

55

EXTRA SLIDES

56

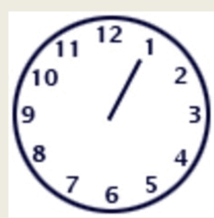
IMPLEMENTING LRU

- 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

57

IMPLEMENTING LRU - 2

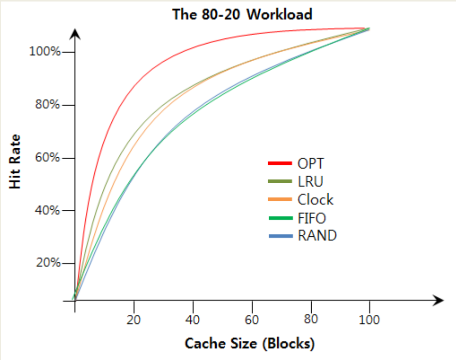
- Harness the Page Table Entry (PTE) Use Bit
- HW sets to 1 when page is used
- OS sets to 0
- Clock algorithm (*approximate LRU*)
  - Refer to pages in a circular list
  - Clock hand points to current page
  - Loops around
    - IF USE_BIT=1 set to USE_BIT = 0
    - IF USE_BIT=0 replace page



58

CLOCK ALGORITHM

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



59

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

60

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

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

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