


TCSS 422: OPERATING SYSTEMS

Translation Lookaside Buffer (TLB), Multi-Level Page Tables



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

May 27, 2021

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

OBJECTIVES – 5/27

- **Questions from 5/25**
- Assignment 2 – May 28
- REVIEW: Memory Segmentation Activity (available in Canvas)
- Tutorial 2 – Pthread, locks, conditions tutorial – June 4
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Final exam – alternate format
- Chapter 18: Introduction to Paging
- 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

May 27, 2021

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

L17.2

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

May 27, 2021	TCSS422: Computer Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.3
--------------	--	-------

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

May 27, 2021	TCSS422: Computer Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.4
--------------	--	-------

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (45 respondents):
 - 1-mostly review, 5-equal new/review, 10-mostly new
 - **Average – 6.35 (↑ - previous 6.30)**

- Please rate the pace of today's class:
 - 1-slow, 5-just right, 10-fast
 - **Average – 5.47 (↓ - previous 5.62)**

May 27, 2021	TCSS422: Computer Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.5
--------------	--	-------

FEEDBACK

May 27, 2021	TCSS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.6
--------------	---	-------

OBJECTIVES – 5/27

- Questions from 5/25
- **Assignment 2 – May 28**
- REVIEW: Memory Segmentation Activity (available in Canvas)
- Tutorial 2 – Pthread, locks, conditions tutorial – June 4
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Final Exam – alternate format
- Chapter 18: Introduction to Paging
- 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

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

OBJECTIVES – 5/27

- Questions from 5/25
- Assignment 2 – May 28
- **REVIEW: Memory Segmentation Activity (available in Canvas)**
- Tutorial 2 – Pthread, locks, conditions tutorial – June 4
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Final Exam – alternate format
- Chapter 18: Introduction to Paging
- 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

May 27, 2021	TCSS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.8
--------------	---	-------

OBJECTIVES – 5/27

- Questions from 5/25
- Assignment 2 – May 28
- REVIEW: Memory Segmentation Activity (available in Canvas)
- **Tutorial 2 – Pthread, locks, conditions tutorial – June 4**
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Final Exam – alternate format
- Chapter 18: Introduction to Paging
- 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

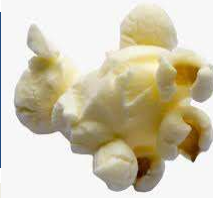
May 27, 2021	TCSS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.9
--------------	---	-------

OBJECTIVES – 5/27

- Questions from 5/25
- Assignment 2 – May 28
- REVIEW: Memory Segmentation Activity (available in Canvas)
- Tutorial 2 – Pthread, locks, conditions tutorial – June 4
- **Assignment 3: (Tutorial) Introduction to Linux Kernel Modules**
- Final Exam – alternate format
- Chapter 18: Introduction to Paging
- 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

May 27, 2021	TCSS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.10
--------------	---	--------

WHAT IS THE KERNEL?



- What is the Linux Kernel?
 - Linux kernel is the core executable engine that facilitates the Linux operating system
- Where is the Linux Kernel?
 - Located under /boot
 - Can have multiple versions on the system
 - Check current version with `uname -a`

```
-rw-r--r-- 1 root root 217414 Apr 14 06:15 config-4.15.0-143-generic
-rw-r--r-- 1 root root 44328834 May 18 06:29 initrd.img-4.15.0-143-generic
-rw----- 1 root root 4081437 Apr 14 06:15 System.map-4.15.0-143-generic
-rw----- 1 root root 8449696 Apr 14 06:18 vmlinuz-4.15.0-143-generic
```

May 27, 2021

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

L17.11

COMPONENTS UNDER /BOOT

```
-rw-r--r-- 1 root root 217414 Apr 14 06:15 config-4.15.0-143-generic
-rw-r--r-- 1 root root 44328834 May 18 06:29 initrd.img-4.15.0-143-generic
-rw----- 1 root root 4081437 Apr 14 06:15 System.map-4.15.0-143-generic
-rw----- 1 root root 8449696 Apr 14 06:18 vmlinuz-4.15.0-143-generic
```

- **config-<ver>**: Kernel configuration
- **initrd-<ver>**: initial root file system that is mounted prior to when the real root file system is available. The initrd is bound to the kernel and loaded as part of the kernel boot procedure.
- **System.map-<ver>**: is a symbol file for the kernel. It lists function entry points and addresses of kernel data structures of a particular build of a kernel.
- **vmlinuz-<ver>**: the Linux kernel executable. The 'z' indicates that it is compressed. An 'x' indicates uncompressed (vmlinux). This can be a symbolic link to the actual kernel

May 27, 2021

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

L17.12

ASSIGNMENT 3: INTRODUCTION TO LINUX KERNEL MODULES

- Linux Kernel Modules are modules of code that are loaded and unloaded into the Linux kernel on demand
- They extend the functionality of the kernel without integrating new code into the kernel which requires recompilation of the full kernel (can be time consuming)
- Kernel modules can be dynamically loaded and removed ***without rebooting*** the system
- `sudo lsmod` – lists actively loaded kernel modules
- **HOW MANY DO YOU HAVE?** - (`sudo lsmod | wc -l`)

May 27, 2021

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

L17.13

ASSIGNMENT 3 – (2): 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

May 27, 2021

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

L17.14

OBJECTIVES – 5/27

- Questions from 5/25
- Assignment 2 – May 28
- REVIEW: Memory Segmentation Activity (available in Canvas)
- Tutorial 2 – Pthread, locks, conditions tutorial – June 4
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- **Final Exam – alternate format**
- Chapter 18: Introduction to Paging
- 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

May 27, 2021

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

L17.15

ALTERNATE FINAL EXAM

- Final Exam category will have two assignments
- Thursday June 8 from 3:40 to 5:40 pm
 - Final Quiz (50 points)
 - SHORT: fewer than half the number of questions as the midterm
 - 1-hour
 - Focus on new content - since the midterm
- Tutorial: Linux File Systems and Disk I/O
 - Available for 1-week ~June 5th to June 11th
 - 50 points
 - Presents new material in a hands-on, interactive format
 - Complete activity and answer questions
 - Individual work

May 27, 2021

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

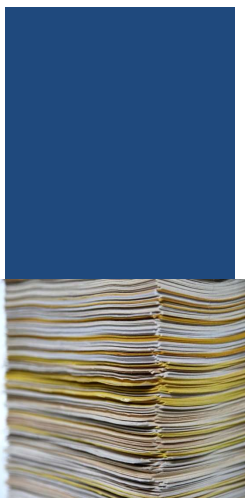
L17.16

OBJECTIVES – 5/27

- Questions from 5/25
- Assignment 2 – May 28
- REVIEW: Memory Segmentation Activity (available in Canvas)
- Tutorial 2 – Pthread, locks, conditions tutorial – June 4
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Final Exam – alternate format
- **Chapter 18: Introduction to Paging**
- 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

May 27, 2021	TCSS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.17
--------------	---	--------

CHAPTER 18: INTRODUCTION TO PAGING



May 27, 2021	TCSS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.18
--------------	---	--------

Consider a 4GB Computer with 4KB (4096 byte) pages. How many pages would fit into physical memory?

$2^{32} / 2^{20} = 2^{12}$ pages

$2^{32} / 2^{12} = 2^{20}$ pages

$2^{32} / 2^{16} = 2^{16}$ pages

$2^{32} / 2^8 = 2^{24}$ pages

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

For the 4GB computer example, how many bits are required for the VPN?

24 VPN bits (indexes
 2^{24} locations)

16 VPN bits (indexes
 2^{16} locations)

20 VPN bits (indexes
 2^{20} locations)

12 VPN bits (indexes
 2^{12} locations)

None of the above

May 27, 2021 TCCS422: Operating Systems [Spring 2021] Start the presentation to see live content. For screen share software, share the entire screen. Get help at pollev.com/app

L17.10

For the 4GB computer example, how many bits are available for page status bits?

32 - 12 VPN bits
= 20 status bits

32 - 24 VPN bits
= 8 status bits

32 - 16 VPN bits
= 16 status bits

32 - 20 VPN bits
= 12 status bits

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

For the 4GB computer, how much space does this page table require? (number of page table entries x size of page table entry)

2^{20} entries x 4b = 4 MB

2^{12} entries x 4b = 16 KB

2^{16} entries x 4b = 256 KB

2^{24} entries x 4b = 64 MB

None of the above

May 27, 2021 TCSS422: Operating Systems [Spring 2021] Start the presentation to see live content. For screen share software, share the entire screen. Get help at pollev.com/app

L17.2
2

For the 4GB computer, how many page tables (for user processes) would fill the entire 4GB of memory?

4 GB / 16 KB = 65,536

4 GB / 64 MB = 256

4GB / 256 KB = 16,384

4GB / 4MB = 1,024

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

PAGING SYSTEM EXAMPLE

- Consider a 4GB Computer:
 - With a 4096-byte page size (4KB)
 - How many pages would fit in physical memory?


- Now consider a page table:
 - For the page table entry, how many bits are required for the VPN?
 - If we assume the use of 4-byte (32 bit) page table entries, how many bits are available for status bits?
 - How much space does this page table require?
of page table entries x size of page table entry
 - How many page tables (for user processes) would fill the entire 4GB of memory?

May 27, 2021	TCSS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.24
--------------	---	--------

OBJECTIVES – 5/27

- Questions from 5/25
- Assignment 2 – May 28
- REVIEW: Memory Segmentation Activity (available in Canvas)
- Tutorial 2 – Pthread, locks, conditions tutorial – June 4
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Final Exam – alternate format
- Chapter 18: Introduction to Paging
- **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

May 27, 2021	TCSS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.25
--------------	---	--------



CHAPTER 19: TRANSLATION LOOKASIDE BUFFER (TLB)

May 27, 2021	TCSS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.26
--------------	---	--------

TRANSLATION LOOKASIDE BUFFER

- Legacy name...
- Better name, “Address Translation Cache”
- TLB is an on CPU cache of address translations
 - virtual → physical memory

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

TRANSLATION LOOKASIDE BUFFER - 2

- Goal:
Reduce access to the page tables
- Example:
50 RAM accesses for first 5 for-loop iterations
- Move lookups from RAM to TLB by caching page table entries

The figure consists of three vertically stacked graphs sharing a common x-axis labeled 'Memory Access' from 0 to 50.

- Top Graph (Page Tables):** Shows access to Page Table[1] (at address 1024) and Page Table[39] (at address 1174). Accesses to Page Table[1] occur at memory accesses 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, and 50. Accesses to Page Table[39] occur at memory accesses 5, 15, 25, 35, and 45.
- Middle Graph (Array):** Shows access to Array[VA] at addresses 40000, 40050, and 40100. Accesses occur at memory accesses 5, 15, 25, 35, and 45.
- Bottom Graph (Code):** Shows access to Code[VA] at addresses 1024, 1074, and 1124. Accesses occur at memory accesses 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, and 50.

May 27, 2021	TCSS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.28
--------------	---	--------

TRANSLATION LOOKASIDE BUFFER (TLB)

- Part of the CPU's Memory Management Unit (MMU)
- Address translation cache

The diagram illustrates the address translation process. On the left, a blue box labeled 'CPU' points to a white box 'Logical Address'. An arrow labeled 'TLB Lookup' points from the 'Logical Address' to a green box 'MMU'. Inside the 'MMU' box, there is a smaller green box 'TLB popular v to p' and a white box 'Page Table all v to p entries'. An arrow labeled 'TLB Hit' points from the 'TLB' box to a white box 'Physical Address'. An arrow labeled 'TLB Miss' points from the 'Page Table' box to the 'Physical Address' box. Below the 'Physical Address' box is a stack of boxes representing 'Physical Memory', labeled 'Page 0', 'Page 1', 'Page 2', '...', and 'Page n'. The entire diagram is captioned 'Address Translation with MMU' and 'Physical Memory'.

May 27, 2021	TCSS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.29
--------------	---	--------

TRANSLATION LOOKASIDE BUFFER (TLB)

- Part of the CPU's Memory Management Unit (MMU)
- Address translation cache

**The TLB is an address translation cache
Different than L1, L2, L3 CPU memory caches**

This diagram is identical to the one in slide L17.29, showing the flow from CPU to Logical Address, through MMU (TLB and Page Table), to Physical Address, and finally to Physical Memory (Page 0 to Page n). The callout box highlights that the TLB is an address translation cache, distinct from L1, L2, or L3 CPU memory caches.

May 27, 2021	TCSS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.30
--------------	---	--------

OBJECTIVES – 5/27

- Questions from 5/25
- Assignment 2 – May 28
- REVIEW: Memory Segmentation Activity (available in Canvas)
- Tutorial 2 – Pthread, locks, conditions tutorial – June 4
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Final Exam – alternate format
- Chapter 18: Introduction to Paging
- 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

May 27, 2021

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

L17.31

TLB BASIC ALGORITHM

- For: array based page table
- Hardware managed TLB

```
1: VPN = (VirtualAddress & VPN_MASK ) >> SHIFT
2: (Success , TlbEntry) = TLB_Lookup(VPN)
3:   if(Success == True){ // TLB Hit
4:     if(CanAccess(TlbEntry.ProtectBits) == True ){
5:       Offset = VirtualAddress & OFFSET_MASK
6:       PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
7:       AccessMemory( PhysAddr )
8:     }else RaiseException( PROTECTION_ERROR)
```

Generate the physical address to access memory

May 27, 2021

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

L17.32

TLB BASIC ALGORITHM - 2

```
11:     else{ //TLB Miss
12:         PTEAddr = PTBR + (VPN * sizeof(PTE))
13:         ➡ PTE = AccessMemory(PTEAddr)
14:         (...) // Check for, and raise exceptions...
15:
16:         ➡ TLB_Insert( VPN , PTE.PFN , PTE.ProtectBits)
17:         ➡ RetryInstruction()
18:     }
19: }
```

Retry the instruction... (requery the TLB)

May 27, 2021

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

L17.33

TLB – ADDRESS TRANSLATION CACHE

- Key detail:
- For a TLB miss, we first access the page table in RAM to populate the TLB... we then requery the TLB
- All address translations go through the TLB

May 27, 2021

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

L17.34

OBJECTIVES – 5/27

- Questions from 5/25
- Assignment 2 – May 28
- REVIEW: Memory Segmentation Activity (available in Canvas)
- Tutorial 2 – Pthread, locks, conditions tutorial – June 4
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Final Exam – alternate format
- Chapter 18: Introduction to Paging
- 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

May 27, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.35
--------------	---	--------

TLB EXAMPLE

```

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

- Example:
- Program address space: 256-byte
 - Addressable using 8 total bits (2^8)
 - 4 bits for the VPN (16 total pages)
- Page size: 16 bytes
 - Offset is addressable using 4-bits
- Store an array: of (10) 4-byte integers

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

May 27, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.36
--------------	---	--------

TLB EXAMPLE - 2

```

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

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

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

May 27, 2021
TCSS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma
L17.37

TLB EXAMPLE - 3

```

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

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

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

May 27, 2021
TCSS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma
L17.38

TLB EXAMPLE - 4

```

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

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

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

May 27, 2021

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

L17.39

OBJECTIVES – 5/27

- Questions from 5/25
- Assignment 2 – May 28
- REVIEW: Memory Segmentation Activity (available in Canvas)
- Tutorial 2 – Pthread, locks, conditions tutorial – June 4
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Final Exam – alternate format
- Chapter 18: Introduction to Paging
- 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

May 27, 2021

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


L17.40

**WE WILL RETURN AT
4:50PM**



May 27, 2021 TCSS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma L17.41

**CHAPTER 20:
PAGING:
SMALLER TABLES**



May 27, 2021 TCSS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma L17.42

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

May 27, 2021

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

L17.43

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

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

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

May 27, 2021

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

L17.44

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

May 27, 2021

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

L17.45

OBJECTIVES – 5/27

- Questions from 5/25
- Assignment 2 – May 28
- REVIEW: Memory Segmentation Activity (available in Canvas)
- Tutorial 2 – Pthread, locks, conditions tutorial – June 4
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Final Exam – alternate format
- Chapter 18: Introduction to Paging
- 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

May 27, 2021

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

L17.46

PAGING: USE LARGER PAGES

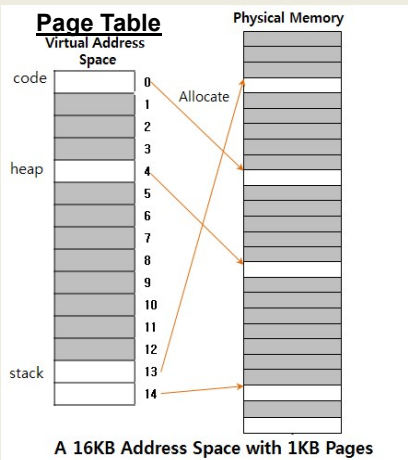
- **Larger pages = 16KB = 2¹⁴**
- **32-bit address space: 2³²**
- **2¹⁸ = 262,144 pages**

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

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

PAGE TABLES: WASTED SPACE

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



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

A Page Table For 16KB Address Space

MULTI-LEVEL PAGE TABLES

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

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

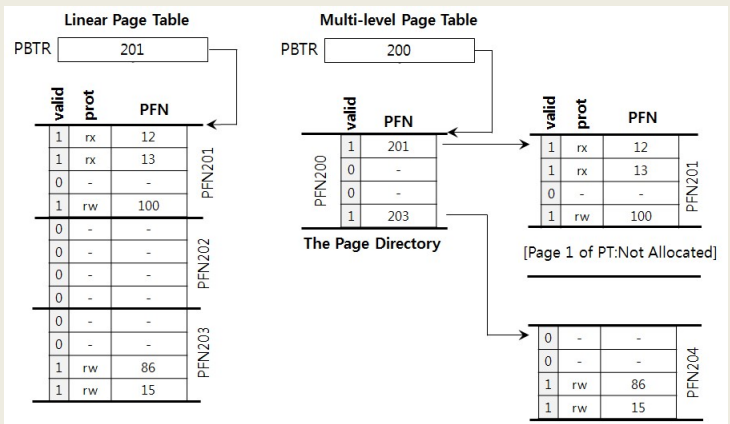
- Often most of the 4MB *per process* page table is empty
- Page table must be placed in 4MB contiguous block of RAM

- **MUST SAVE MEMORY!**

May 27, 2021	TCSS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.51
--------------	---	--------

MULTI-LEVEL PAGE TABLES - 2

- Add level of indirection, the “page directory”



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

May 27, 2021	TCSS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.52
--------------	---	--------

MULTI-LEVEL PAGE TABLES - 2

- Add level of indirection, the “page directory”

Linear Page Table

PBTR 201

Multi-level Page Table

PBTR 200

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

0	-	-	PFN203
0	-	-	
1	rw	86	
1	rw	15	

0	-	-	PFN204
0	-	-	
1	rw	86	
1	rw	15	

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

May 27, 2021

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

L17.53

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

May 27, 2021

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

L17.54

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					

May 27, 2021

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

L17.55

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!**

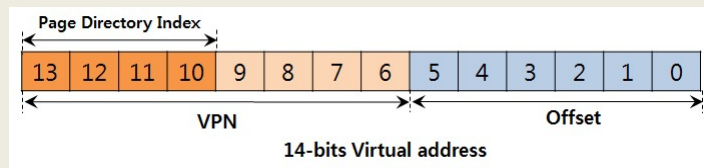
May 27, 2021

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

L17.56

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



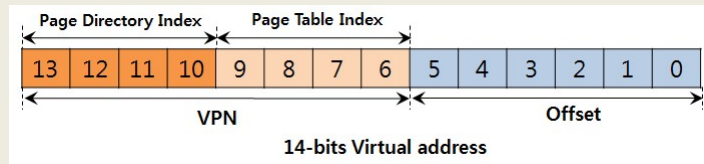
May 27, 2021

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

L17.57

PAGE TABLE INDEX

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



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

May 27, 2021

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

L17.58

EXAMPLE - 3

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

May 27, 2021

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

L17.59

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

May 27, 2021

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

L17.60

OBJECTIVES – 5/27

- Questions from 5/25
- Assignment 2 – May 28
- REVIEW: Memory Segmentation Activity (available in Canvas)
- Tutorial 2 – Pthread, locks, conditions tutorial – June 4
- Assignment 3: (Tutorial) Introduction to Linux Kernel Modules
- Final Exam – alternate format
- Chapter 18: Introduction to Paging
- 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

May 27, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.61
--------------	---	--------

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 → $\log_2 128 = 7$

May 27, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L17.62
--------------	---	--------

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

$\overbrace{\text{Page Directory Index}}^{\text{VPN}} \quad \overbrace{\text{Page Table Index}}^{\text{offset}}$

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

May 27, 2021

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

L17.63

MORE THAN TWO LEVELS - 3

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

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

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

May 27, 2021

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

L17.63

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	→ $\log_2 128 = 7$

May 27, 2021
TCSS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma
L17.65

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% !!!

May 27, 2021
TCSS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma
L17.66

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;
```

May 27, 2021

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

L17.67

ADDRESS TRANSLATION - 2

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

pgd_offset():

Takes a vpage address and the mm_struct for the process, returns the PGD entry that covers the requested address...

p4d/pud/pmd_offset():

Takes a vpage address and the pgd/p4d/pud entry and returns the relevant p4d/pud/pmd.

pte_unmap()

release temporary kernel mapping for the page table entry

May 27, 2021

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

L17.68

INVERTED PAGE TABLES



- Keep a single page table for each physical page of memory
- Consider 4GB physical memory
- Using 4KB pages, page table requires 4MB to map all of RAM
- Page table stores
 - Which process uses each page
 - Which process virtual page (from process virtual address space) maps to the physical page
- All processes share the same page table for memory mapping, kernel must isolate all use of the shared structure
- Finding process memory pages requires search of 2^{20} pages
- Hash table: can index memory and speed lookups

May 27, 2021

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

L17.69

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?

May 27, 2021

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

L17.70

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)?

May 27, 2021

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

L17.71

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

May 27, 2021

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

L17.72

MULTI LEVEL PAGE TABLE EXAMPLE - 4

- **(#9)** Using a single page directory entry (PDE) pointing to a single page table (PT), if all of the slots of the page table (PT) are in use, what is the total amount of memory a two-level page table scheme can address?
- **(#10)** And finally, for this example, as a percentage (%), how much memory does the 2-level page table scheme consume compared to the 1-level scheme?
- **HINT:** two-level memory use / one-level memory use

May 27, 2021

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

L17.73

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

May 27, 2021

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

L17.74

