


TCSS 422: OPERATING SYSTEMS

Memory Virtualization II:
Memory Segments,
Free Space Management,
Introduction to Paging,
Translation Lookaside Buffer

Wes J. Lloyd

School of Engineering and Technology

University of Washington - Tacoma



May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington

Tacoma

1

TCSS 422 – OFFICE HRS – SPRING 2025

■ Office Hours plan for Spring (by Zoom):

■ Monday 11:30am - 12:30p GTA Xinghan

■ Tuesday 11:30am - 12:30p GTA Xinghan

■ Wednesday 11:00am - 12:00p Instructor Wes

■ Friday 12:00pm - 1:00p Instructor Wes or GTA Xinghan

■ THIS FRIDAY: Xinghan

■ Instructor is available immediately after class today via Zoom for questions

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.2

2

OBJECTIVES – 5/22

Questions from 5/20

Assignment 2 - June 5 AOE

Quiz 3 – Synchronized Array - May 26 AOE

Tutorial 2 – Pthread/locks/conditions tutorial 5/29 AOE

Assignment 3 (as a Tutorial) to be posted...

Chapter 16: Segmentation

Chapter 17: Free Space Management

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 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.3

3

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 - Background survey

May 22, 2025

TCSS422: Computer Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.4

4

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:

12345678910

Mostly Review To MeEqual New and ReviewMostly New to Me

Question 2

0.5 pts

Please rate the pace of today's class:

12345678910

SlowJust RightFast

May 22, 2025

TCSS422: Computer Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.5

5

MATERIAL / PACE

Please classify your perspective on material covered in today's class (36 of 63 respondents – 57.1% @ 11:22pm):

1-mostly review, 5-equal new/review, 10-mostly new

Average – 5.96 (↓ - previous 6.51)

Please rate the pace of today's class:

1-slow, 5-just right, 10-fast

Average – 4.55 (↓ - previous 5.00)

May 22, 2025

TCSS422: Computer Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.6

6

FEEDBACK FROM 5/20

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.7

7

OBJECTIVES – 5/22

- Questions from 5/20
- Assignment 2 - June 5 AOE
- Quiz 3 – Synchronized Array - May 26 AOE
- Tutorial 2 – Pthread/locks/conditions tutorial 5/29 AOE
- Assignment 3 (as a Tutorial) to be posted...
- Chapter 16: Segmentation
- Chapter 17: Free Space Management
- 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 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.8

8

OBJECTIVES – 5/22

- Questions from 5/20
- Assignment 2 - June 5 AOE
- Quiz 3 – Synchronized Array - May 26 AOE
- Tutorial 2 – Pthread/locks/conditions tutorial 5/29 AOE
- Assignment 3 (as a Tutorial) to be posted...
- Chapter 16: Segmentation
- Chapter 17: Free Space Management
- 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 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.9

9

OBJECTIVES – 5/22

- Questions from 5/20
- Assignment 2 - June 5 AOE
- Quiz 3 – Synchronized Array - May 26 AOE
- Tutorial 2 – Pthread/locks/conditions tutorial 5/29 AOE
- Assignment 3 (as a Tutorial) to be posted...
- Chapter 16: Segmentation
- Chapter 17: Free Space Management
- 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 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.10

10

OBJECTIVES – 5/22

- Questions from 5/20
- Assignment 2 - June 5 AOE
- Quiz 3 – Synchronized Array - May 26 AOE
- Tutorial 2 – Pthread/locks/conditions tutorial 5/29 AOE
- **Assignment 3 (as a Tutorial) to be posted...**
- Chapter 16: Segmentation
- Chapter 17: Free Space Management
- 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 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.11

11

CATCH UP FROM LECTURE 14

- Switch to Lecture 14 Slides
- Skip Slides 13.47 – 13.48
- Slides L14.24 to L14.37
(Chapter 14 – The Memory API)

May 20, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma


L14.12

12

CHAPTER 15: ADDRESS TRANSLATION

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma



L15.13

13

OBJECTIVES – 5/18

■ Chapter 15: Address translation

■ Base and bounds

■ HW and OS Support

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.14

14

ADDRESS TRANSLATION

- **64KB Address space example**
- **Translation: mapping virtual to physical**

0KB
16KB

0KB
16KB
32KB
48KB
64KB

Virtual mapping
Address Space
Physical Memory

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.15

15

BASE AND BOUNDS

- **Dynamic relocation**
- **Two registers base & bounds: on the CPU**
- **OS places program in memory**
- **Sets base register**

$$physical\ address = virtual\ address + base$$

- **Bounds register**
 - **Stores size of program address space (16KB)**
- **OS verifies that every address:**

$$0 \leq virtual\ address < bounds$$

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.16

16

INSTRUCTION EXAMPLE

128 : movl 0x0(%ebx), %eax

- Base = 32768
- Bounds =16384
- Fetch instruction at 128 (virt addr) ↑
 - Phy addr = virt addr + base reg
 - 32896 = 128 + 32768 (base)
- Execute instruction
 - Load from address (var x is @ 15kb=15360)
 - 48128 = 15360 + 32768 (base) -- found x...
- Bounds register: terminate process if
 - ACCESS VIOLATION: Virtual address > bounds reg

physical address = virtual address + base

0KB128movl 0x0(%ebx), %eax

1KB132addl 0x05, %eax

1KB135movl %eax, 0x0(%ebx)

2KB

3KBProgram Code

4KB

Heap

↓

heap

(free)

↑

stack

14KB

15KB3000Int x

16KBStack

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.17

17

MEMORY MANAGEMENT UNIT

- MMU
 - Portion of the CPU dedicated to address translation
 - Contains base & bounds registers
- Base & Bounds Example:
 - Consider address translation
 - 4 KB (4096 bytes) address space, loaded at 16 KB physical location

Virtual Address	Physical Address
0	16384
1024	17408
3000	19384
4400	20784 (out of bounds)

FAULT

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.18

18

Slides by Wes J. Lloyd

L15.9

DYNAMIC RELOCATION OF PROGRAMS

- Hardware requirements:

Requirements	HW support
Privileged mode	CPU modes: kernel, user
Base / bounds registers	Registers to support address translation
Translate virtual addr; check if in bounds	Translation circuitry, check limits
Privileged instruction(s) to update base / bounds regs	Instructions for modifying base/bound registers
Privileged instruction(s) to register exception handlers	Set code pointers to OS code to handle faults
Ability to raise exceptions	For out-of-bounds memory access, or attempts to access privileged instr.

May 22, 2025

TCCS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.19

19

OS SUPPORT FOR MEMORY VIRTUALIZATION

- For base and bounds OS support required
 - When process starts running
 - Allocate address space in physical memory
 - When a process is terminated
 - Reclaiming memory for use
 - When context switch occurs
 - Saving and storing the base-bounds pair
 - Exception handlers
 - Function pointers set at OS boot time

May 22, 2025

TCCS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.20

20

OS: WHEN PROCESS STARTS RUNNING

■ OS searches for free space for new process

■ Free list: data structure that tracks available memory slots

The OS lookup the free list

Free list

16KB

48KB

0KB

16KB

32KB

48KB

64KB

Operating System

(not in use)

Code

Heap

(allocated but not in use)

Stack

(not in use)

Physical Memory

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.21

21

OS: WHEN PROCESS IS TERMINATED

■ OS places memory back on the free list

Free list

16KB

32KB

48KB

0KB

16KB

32KB

48KB

64KB

Operating System

(not in use)

(not in use)

(not in use)

Physical Memory

Free list

16KB

32KB

48KB

0KB

16KB

32KB

48KB

64KB

Operating System

(not in use)

(not in use)

(not in use)

Physical Memory

May 22, 2025

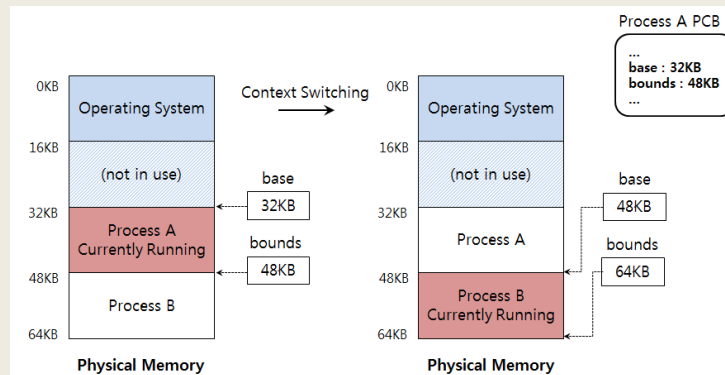
TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.22

22

OS: WHEN CONTEXT SWITCH OCCURS

- OS must save base and bounds registers
 - Saved to the Process Control Block PCB (`task_struct` in Linux)



May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.23

23

DYNAMIC RELOCATION

- OS can move process data when not running
 - OS un-schedules process from scheduler
 - OS copies address space from current to new location
 - OS updates PCB (base and bounds registers)
 - OS reschedules process
- When process runs new base register is restored to CPU
- Process doesn't know it was even moved!**

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.24

24

Consider a 64KB computer the loads a program. The BASE register is set to 32768, and the BOUNDS register is set to 4096. What is the physical memory address translation for a virtual address of 6000 ?

34768

38768

32769

36864

Out of bounds

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

25

**WE WILL RETURN AT
4:55PM**



May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington -

coma

L15.26

26

OBJECTIVES – 5/22

- Questions from 5/20
- Assignment 2 - June 5 AOE
- Quiz 3 – Synchronized Array - May 26 AOE
- Tutorial 2 – Pthread/locks/conditions tutorial 5/29 AOE
- Assignment 3 (as a Tutorial) to be posted...
- **Chapter 16: Segmentation**
- Chapter 17: Free Space Management
- 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 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.27

27

CHAPTER 16:
SEGMENTATION



May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.28

28

BASE AND BOUNDS INEFFICIENCIES

- Address space
 - Contains significant unused memory
 - Is relatively large
 - Preallocates space to handle stack/heap growth
- Large address spaces
 - Hard to fit in memory
- How can these issues be addressed?

The diagram illustrates memory layout inefficiencies. It shows a vertical stack of memory segments: Program Code (0KB to 3KB), Heap (4KB to 6KB), a large free space (6KB to 14KB), and Stack (14KB to 16KB). Arrows indicate the growth of the Heap and Stack towards each other, leaving a significant gap of unused memory.

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.29

29

MULTIPLE SEGMENTS

- Memory segmentation
- Manage the address space as (3) separate segments
 - Each is a contiguous address space
 - Provides logically separate segments for: code, stack, heap
- Each segment can placed separately
- Track base and bounds for each segment (registers)

The diagram illustrates memory segmentation, showing how the address space is divided into separate, contiguous segments for code, stack, and heap, which can be placed separately in memory.

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.30

30

SEGMENTS IN MEMORY

Consider 3 segments:

0KB

Operating System

16KB

(not in use)

Stack

(not in use)

32KB

Code

Heap

48KB

(not in use)

64KB

Physical Memory

Much smaller

↓

Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.31

31

ADDRESS TRANSLATION: CODE SEGMENT

$$\text{physical address} = \text{offset} + \text{base}$$

Code segment - physically starts at 32KB (base)

Starts at "0" in virtual address space

Segment	Base	Size	
Code			16KB

0KB

100 Instruction

2KB Program

4KB

Virtual Address Space

(not in use)

Physical Address Space

Bounds check:

Is virtual address within 2KB address space?

or 32868 desired address

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.32

32

Slides by Wes J. Lloyd

L15.16

ADDRESS TRANSLATION: HEAP

Virtual address + base is not the correct physical address.

- Heap starts at virtual address 4096
- The data is at 4200
- Offset= 4200 - 4096 = 104 (virt addr - virt heap start)
- Physical address = 104 + 34816 (offset + heap base)

Segment	Base	Size
Heap	34K	2K

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.33

33

SEGMENTATION FAULT

- Access beyond the address space
- Heap starts at virtual address: 4096
- Data pointer is to 7KB (7168)
- Is data pointer valid?

- Heap starts at 4096 + 2 KB seg size = 6144
- Offset= 7168 > 4096 + 2048 (6144)

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.34

34

SEGMENT REGISTERS

- Used to dereference memory during translation

13 12 11 10 9 8 7 6 5 4 3 2 1 0

SegmentOffset

- First two bits identify segment type
- Remaining bits identify memory offset
- Example: virtual heap address 4200 (01000001101000)

13 12 11 10 9 8 7 6 5 4 3 2 1 0

0 1 0 0 0 0 0 0 1 1 0 1 0 0 0

SegmentOffset

Segment	bits
Code	00
Heap	01
Stack	10
-	11

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.35

35

SEGMENTATION DEREFERENCE

```
1 // get top 2 bits of 14-bit VA
2 Segment = (VirtualAddress & SEG_MASK) >> SEG_SHIFT
3 // now get offset
4 Offset = VirtualAddress & OFFSET_MASK
5 if (Offset >= Bounds[Segment])
6     RaiseException(PROTECTION_FAULT)
7 else
8     PhysAddr = Base[Segment] + Offset
9     Register = AccessMemory(PhysAddr)
```

- VIRTUAL ADDRESS = 01000001101000 (on heap)
- SEG_MASK = 0x3000 (11000000000000)
- SEG_SHIFT = 01 → heap (mask gives us segment code)
- OFFSET_MASK = 0xFFF (00111111111111)
- OFFSET = 000001101000 = 104 (isolates segment offset)
- OFFSET < BOUNDS : 104 < 2048

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.36

36

STACK SEGMENT

- Stack grows backwards (FILO)
- Requires hardware support:
- Direction bit: tracks direction segment grows

26KB

(not in use)

↑

Stack

28KB

(not in use)

Physical Memory

Segment Register(with Negative-Growth Support)

Segment	Base	Size	Grows	Positive?
Code	32K	2K		1
Heap	34K	2K		1
Stack	28K	2K		0

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.37

37

SHARED CODE SEGMENTS

- Code sharing: enabled with HW support
- Supports storing shared libraries in memory only once
- DLL: dynamic linked library
- .so (linux): shared object in Linux (under /usr/lib)
- Many programs can access them
- Protection bits: track permissions to segment

Segment Register Values(with Protection)

Segment	Base	Size	Grows	Positive?	Protection
Code	32K	2K		1	Read-Execute
Heap	34K	2K		1	Read-Write
Stack	28K	2K		0	Read-Write

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.38

38

Consider a program with 2KB of code, a 1 KB stack, and a 2 KB heap. This program runs on a 64 KB computer that manages memory with 4 kb segments. If the computer is empty and segments were allocated as: code, stack, heap, how large can the heap grow to?

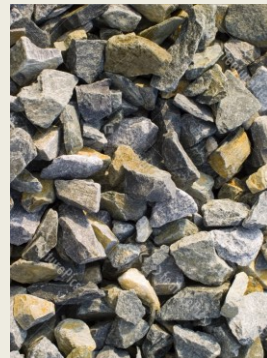
32 KB
56 KB
24 KB
4 KB
0 KB

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

39

SEGMENTATION GRANULARITY

- Coarse-grained
- Manage memory as large purpose based segments:
 - Code segment
 - Heap segment
 - Stack segment



May 22, 2025


TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.40

40

SEGMENTATION GRANULARITY - 2

- Fine-grained
- Manage memory as list of segments
- Code, heap, stack segments composed of multiple smaller segments
- Segment table
 - On early systems
 - Stored in memory
 - Tracked large number of segments



May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.41

41

MEMORY FRAGMENTATION

- Consider how much free space?
- We'll say about 24 KB
- Request arrives to allocate a 20 KB heap segment
- Can we fulfil the request for 20 KB of contiguous memory?

Not compacted	
0KB	
8KB	Operating System
16KB	(not in use)
24KB	Allocated
32KB	(not in use)
40KB	Allocated
48KB	(not in use)
56KB	Allocated
64KB	

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.42

42

COMPACTION

- Supports rearranging memory
- Can we fulfil the request for 20 KB of contiguous memory?
- **Drawback: Compaction is slow**
 - Rearranging memory is time consuming
 - 64KB is fast
 - 4GB+ ... slow
- **Algorithms:**
 - Best fit: keep list of free spaces, allocate the most snug segment for the request
 - Others: worst fit, first fit... (in future chapters)

0KB	Operating System
8KB	
16KB	Allocated
24KB	
32KB	
40KB	
48KB	(not in use)
56KB	
64KB	

May 22, 2025	TCCS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma	L15.43
--------------	---	--------

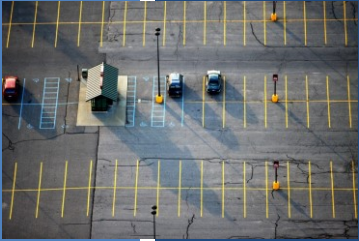
43

OBJECTIVES – 5/22

- Questions from 5/20
- Assignment 2 - June 5 AOE
- Quiz 3 – Synchronized Array - May 26 AOE
- Tutorial 2 – Pthread/locks/conditions tutorial 5/29 AOE
- Assignment 3 (as a Tutorial) to be posted...
- Chapter 16: Segmentation
- **Chapter 17: Free Space Management**
- 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 22, 2025	TCCS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma	L15.44
--------------	---	--------

44



CHAPTER 17: FREE SPACE MANAGEMENT

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.45

45

OBJECTIVES – 5/22

- **Chapter 17: Free Space Management**
 - Fragmentation, Splitting, coalescing
 - The Free List
 - Memory Allocation Strategies

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.46

46

FREE SPACE MANAGEMENT

- How should free space be managed, when satisfying variable-sized requests?
- What strategies can be used to minimize fragmentation?
- What are the time and space overheads of alternate approaches?

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.47

47

FREE SPACE MANAGEMENT

- Management of memory using
 - Only fixed-sized units
 - Easy: keep a list
 - Memory request → return first free entry
 - Simple search
 - With variable sized units
 - More challenging
 - Results from variable sized malloc requests
 - Leads to fragmentation

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.48

48

FRAGMENTATION

- Consider a 30-byte heap

30-byte heap:

free	used	free	
0	10	20	30

- Request for 15-bytes

free list: head →

addr:0	len:10
--------	--------

 →

addr:20	len:10
---------	--------

 → NULL

- Free space: 20 bytes
- No available contiguous chunk → return NULL

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.49

49

FRAGMENTATION - 2

- External: OS can compact
 - Example: Client asks for 100 bytes: malloc(100)
 - OS: No 100 byte contiguous chunk is available: returns NULL
 - Memory is externally fragmented - - Compaction can fix!
- Internal: lost space – OS can't compact
 - OS returns memory units that are too large
 - Example: Client asks for 100 bytes: malloc(100)
 - OS: Returns 125 byte chunk
 - Fragmentation is *in* the allocated chunk
 - Memory is lost, and unaccounted for – can't compact

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.50

50

ALLOCATION STRATEGY: SPLITTING

- Request for 1 byte of memory: `malloc(1)`

30-byte heap:

free	used	free	
0	10	20	30

free list: head →

addr:0	len:10
--------	--------

 →

addr:20	len:10
---------	--------

 → NULL

- OS locates a free chunk to satisfy request
- Splits chunk into two, returns first chunk

30-byte heap:

free	used	free	
0	10	21	30

free list: head →

addr:0	len:10
--------	--------

 →

addr:21	len:9
---------	-------

 → NULL

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.51

51

ALLOCATION STRATEGY: COALESCING

- Consider 30-byte heap
- `Free()` frees all 10 bytes segments (list of 3-free 10-byte chunks)

head →

addr:10	len:10
---------	--------

 →

addr:0	len:10
--------	--------

 →

addr:20	len:10
---------	--------

 → NULL

- Request arrives: `malloc(30)`
- SPLIT DOES NOT WORK** - no contiguous 30-byte chunk exists!
- Coalescing regroups chunks into contiguous chunk

head →

addr:0	len:30
--------	--------

 → NULL

- Allocation can now proceed
- Coalescing is defragmentation of the free space list

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.52

52

MEMORY HEADERS

- `free(void *ptr)`: Does not require a size parameter
- *How does the OS know how much memory to free?*
- Header block
 - Small descriptive block of memory at start of chunk

The diagram shows a memory layout. A pointer labeled 'ptr' points to the start of a rectangular block. This block is divided into two horizontal sections. The top section is labeled 'The header used by malloc library'. The bottom section is labeled 'The 20 bytes returned to caller'. A bracket on the right side of the entire block is labeled 'An Allocated Region Plus Header'.

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.53

53

MEMORY HEADERS - 2

The diagram illustrates the structure of a memory header. On the left, a pointer 'hptr' points to a header block, and a pointer 'ptr' points to the data region. The header block contains two fields: 'size: 20' and 'magic: 1234567'. The data region is labeled 'The 20 bytes returned to caller'. Below the header, the text 'Specific Contents Of The Header' is written. To the right, a C struct definition is shown:

```
typedef struct __header_t {
    int size;
    int magic;
} header_t;
```

 Below the struct definition, the text 'A Simple Header' is written.

- Contains size
- Pointers: for faster memory access
- Magic number: integrity checking

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.54

54

MEMORY HEADERS - 3

- Size of memory chunk is:
 - Header size + user malloc size
 - N bytes + sizeof(header)
- Easy to determine address of header

```
void free(void *ptr) {  
    header_t *hptr = (void *)ptr - sizeof(header_t);  
}
```

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.55

55

THE FREE LIST

- Simple free list struct
- Use mmap to create free list
- 4kb heap, 4 byte header, one contiguous free chunk

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

```
// mmap() returns a pointer to a chunk of free space  
node_t *head = mmap(NULL, 4096, PROT_READ|PROT_WRITE,  
                    MAP_ANON|MAP_PRIVATE, -1, 0);  
head->size = 4096 - sizeof(node_t);  
head->next = NULL;
```

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.56

56

FREE LIST - 2

■ Create and initialize free-list “heap”

```
// mmap() returns a pointer to a chunk of free space
node_t *head = mmap(NULL, 4096, PROT_READ|PROT_WRITE,
                    MAP_ANON|MAP_PRIVATE, -1, 0);
head->size = 4096 - sizeof(node_t);
head->next = NULL;
```

■ Heap layout:

head →

size: 4088

next: 0

...

[virtual address: 16KB]
header: size field

header: next field(NULL is 0)

the rest of the 4KB chunk

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.57

57

FREE LIST: MALLOC() CALL

■ Consider a request for a 100 bytes: malloc(100)

■ Header block requires 8 bytes

- 4 bytes for size, 4 bytes for magic number

■ Split the heap – header goes with each block

A 4KB Heap With One Free Chunk

A Heap : After One Allocation

head →

size: 4088

next: 0

...

the rest of the 4KB chunk

ptr →

size: 100

magic: 1234567

First block is used

head →

size: 3980

next: 0

...

the 100 bytes now allocated

the free 3980 byte chunk

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.58

58

FREE LIST: FREE() CALL

■ Addresses of chunks

■ Start=16384
+ 108 (end of 1st chunk)
+ 108 (end of 2nd chunk)
+ 108 (end of 3rd chunk)
= 16708

8 bytes header

size: 100
magic: 1234567

...

size: 100
magic: 1234567

sptr → **Free this block**

size: 100
magic: 1234567

...

head → size: 3764
next: 0

...

The free 3764-byte chunk

Free Space With Three Chunks Allocated

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.59

59

FREE LIST:
FREE() CHUNK #2

■ Free(sptr)

■ Our 3 chunks start at 16 KB (@ 16,384 bytes)

■ Free chunk #2 - sptr

■ Sptr = 16500
▪ addr – sizeof(node_t)

■ Actual start of chunk #2
▪ 16492

head

size: 100
magic: 1234567

...

size: 100
next: 16708

sptr → **Block Now Free**

size: 100
magic: 1234567

...

size: 3764
next: 0

...

The free 3764-byte chunk

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.60

60

FREE LIST- FREE ALL CHUNKS

- Now free remaining chunks:
- Free(16392)
- Free(16608)
- Walk back 8 bytes for actual start of chunk
- External fragmentation
- Free chunk pointers out of order
- Coalescing of next pointers is needed

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.61

61

GROWING THE HEAP

- Start with small sized heap
- Request more memory when full
- sbrk(), brk()

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.62

62

MEMORY ALLOCATION STRATEGIES

■ **Best fit**

■ Traverse free list

■ Identify all candidate free chunks

■ Note which is smallest (has best fit)

■ When splitting, “leftover” pieces are small (and potentially less useful -- fragmented)

■ **Worst fit**

■ Traverse free list

■ Identify largest free chunk

■ Split largest free chunk, leaving a still large free chunk

May 22, 2025

TCCS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.63

63

EXAMPLES

■ **Allocation request for 15 bytes**

head → 10 → 30 → 20 → NULL

■ **Result of Best Fit**

head → 10 → 30 → 5 → NULL

■ **Result of Worst Fit**

head → 10 → 15 → 20 → NULL

May 22, 2025

TCCS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.64

64

MEMORY ALLOCATION STRATEGIES - 2

■ First fit

- Start search at beginning of free list
- Find first chunk large enough for request
- Split chunk, returning a “fit” chunk, saving the remainder
- Avoids full free list traversal of best and worst fit

■ Next fit

- Similar to first fit, but start search at last search location
- Maintain a pointer that “cycles” through the list
- Helps balance chunk distribution vs. first fit
- Find first chunk, that is large enough for the request, and split
- Avoids full free list traversal

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.65

65

Which memory allocation strategy is more likely to distribute free chunks closer together which could help when coalescing the free space list?

Best Fit

Worst Fit

First Fit

None of the above

All of the above

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

66

SEGREGATED LISTS

- For popular sized requests
e.g. for kernel objects such as locks, inodes, etc.
- Manage as segregated free lists
- Provide object caches: stores pre-initialized objects
- How much memory should be dedicated for specialized requests (object caches)?
- If a given cache is low in memory, can request “*slabs*” of memory from the general allocator for caches.
- General allocator will reclaim slabs when not used

May 22, 2025

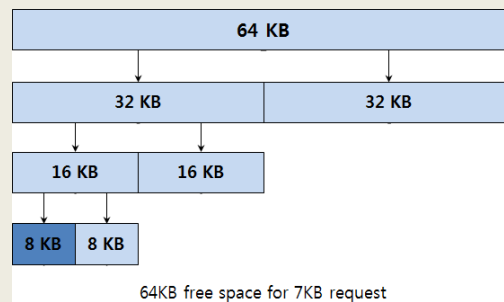
TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.67

67

BINARY BUDDY ALLOCATION

- Binary buddy allocation
 - Divides free space by two to find a block that is big enough to accommodate the request; the next split is too small...
- Consider a 7KB request



May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.68

68

BINARY BUDDY ALLOCATION - 2

- Binary Buddy Allocation: suffers from internal fragmentation
- Allocated fragments, typically too large
- Coalescing is simple
 - Two adjacent blocks are promoted up

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.69

69

A computer system manages program memory using three separate segments for code, stack, and the heap. The codesize of a program is 1KB but the minimal segment available is 16KB. This is an example of:

External fragmentation
Binary buddy allocation
Internal fragmentation
Coalescing
Splitting

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

70

A request is made to store 1 byte. For this scenario, which memory allocation strategy will always locate memory the fastest?

Best fit

Worst fit

Next fit

None of the above

All of the above

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

71


OBJECTIVES – 5/22

- Questions from 5/20
- Assignment 2 - June 5 AOE
- Quiz 3 – Synchronized Array - May 26 AOE
- Tutorial 2 – Pthread/locks/conditions tutorial 5/29 AOE
- Assignment 3 (as a Tutorial) to be posted...
- Chapter 16: Segmentation
- Chapter 17: Free Space Management
- **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 22, 2025	TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma	L15.72
--------------	---	--------

72

CHAPTER 18: INTRODUCTION TO PAGING



May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.73

73

PAGING

- Split up address space of process into fixed sized pieces called **pages**
- Alternative to variable sized pieces (Segmentation) which suffers from significant fragmentation
- Physical memory is split up into an array of fixed-size slots called **page frames**.
- Each process has a **page table** which translates virtual addresses to physical addresses

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.74

74

ADVANTAGES OF PAGING

Flexibility

Abstracts the process address space into pages

No need to track direction of HEAP / STACK growth

- Just add more pages...

No need to store unused space

- As with segments...

Simplicity

Pages and page frames are the same size

Easy to allocate and keep a free list of pages

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.75

75

PAGING: EXAMPLE

Consider a 128 byte (2^7) address space with 16-byte (2^4) pages

Consider a 64-byte (2^6) program address space

0
16
32
48
64

(page 0 of the address space)
(page 1)
(page 2)
(page 3)

A Simple 64-byte Address Space

0
16
32
48
64
80
96
112
128

reserved for OS

(unused)

page 3 of AS

page 0 of AS

(unused)

page 2 of AS

(unused)

page 1 of AS

page frame 0 of physical memory
page frame 1
page frame 2
page frame 3
page frame 4
page frame 5
page frame 6
page frame 7

64-Byte Address Space Placed In Physical Memory

Page Table:
VP0 → PF3
VP1 → PF7
VP2 → PF5
VP3 → PF2

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.76

76

Slides by Wes J. Lloyd

L15.38

PAGING: ADDRESS TRANSLATION

- **PAGE:** Has two address components
 - **VPN:** Virtual Page Number (serves as the page ID)
 - **Offset:** Offset within a Page (indexes any byte in the page)

VPN

offset

Va5

Va4

Va3

Va2

Va1

Va0

VPN

offset

0

1

0

1

0

1

Example:

Page Size: 16-bytes (2^4),

Program Address Space: 64-bytes (2^6)

Here program can have just four pages...

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.77

77

EXAMPLE:
PAGING ADDRESS TRANSLATION

- Consider a 64-byte (2^6) program address space (4 pages $\rightarrow 2^2$)
- Stored in 128-byte (2^7) physical memory (8 frames $\rightarrow 2^3$)

- **Offset is preserved**
 - 4 bits indexes any byte
 - Page size is 16 bytes (2^4)
- **Page table** translates a Virtual Page Number (VPN) to a Physical Frame Number (PFN)

Page Table:
VP0 \rightarrow PF3
VP1 \rightarrow PF7
VP2 \rightarrow PF5
VP3 \rightarrow PF2

Virtual Address

VPN

offset

0

1

0

1

0

1

Address Translation

Physical Address

1

1

1

0

1

0

1

PFN

offset

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.78

78

PAGING DESIGN QUESTIONS

- (1) Where are page tables stored?
- (2) What are the typical contents of the page table?
- (3) How big are page tables?
- (4) Does paging make the system too slow?

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.79

79

(1) WHERE ARE PAGE TABLES STORED?

- Example:
 - Consider a 32-bit process address space ($4\text{GB}=2^{32}$ bytes)
 - With 4 KB pages ($4\text{KB}=2^{12}$ bytes)
 - 20 bits for VPN (2^{20} pages)
 - 12 bits for the page offset (2^{12} unique bytes in a page)
- Page tables for each process are stored in RAM
 - Support potential storage of 2^{20} translations
= 1,048,576 pages per process
 - Each page has a page table entry size of 4 bytes

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.80

80

PAGE TABLE EXAMPLE

- With 2^{20} slots in our page table for a single process
- Each slot (i.e. entry) dereferences a VPN
- Each entry provides a physical frame number
- Each entry requires 4 bytes (32 bits)
 - 20 for the PFN on a 4GB system with 4KB pages
 - 12 for the offset which is preserved
 - (note we have no status bits, so this is unrealistically small)
- How much memory is required to store the page table for 1 process?
 - Hint: # of entries x space per entry
 - 4,194,304 bytes (or 4MB) to index one process

VPN ₀
VPN ₁
VPN ₂
...
...
VPN ₁₀₄₈₅₇₆

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.81

81

NOW FOR AN ENTIRE OS

- If 4 MB is required to store one process
- Consider how much memory is required for an entire OS?
 - With for example 100 processes...
- Page table memory requirement is now 4MB x 100 = 400MB
- If computer has 4GB memory (maximum for 32-bits), the page table consumes 10% of memory

400 MB / 4000 GB

- Is this efficient?

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.82

82

(2) WHAT'S ACTUALLY IN THE PAGE TABLE

- Page table is data structure used to map virtual page numbers (VPN) to the physical address (Physical Frame Number PFN)
 - Linear page table → simple array
- Page-table entry
 - 32 bits for capturing state

31	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
PFN																						G	PAT	D	A	PCD	PWT	U/S	R/W	P	

An x86 Page Table Entry(PTE)

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.83

83

PAGE TABLE ENTRY

- P: present
- R/W: read/write bit
- U/S: supervisor
- A: accessed bit
- D: dirty bit
- PFN: the page frame number

31	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
PFN																						G	PAT	D	A	PCD	PWT	U/S	R/W	P	

An x86 Page Table Entry(PTE)

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.84

84

PAGE TABLE ENTRY - 2

- Common flags:
- Valid Bit:** Indicating whether the particular translation is valid.
- Protection Bit:** Indicating whether the page could be read from, written to, or executed from
- Present Bit:** Indicating whether this page is in physical memory or on disk(swapped out)
- Dirty Bit:** Indicating whether the page has been modified since it was brought into memory
- Reference Bit(Accessed Bit):** Indicating that a page has been accessed

May 22, 2025	TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma	L15.85
--------------	---	--------

85

(3) HOW BIG ARE PAGE TABLES?

- Page tables are too big to store on the CPU
- Page tables are stored using physical memory
- Paging supports efficiently storing a sparsely populated address space
 - Reduced memory requirement
Compared to base and bounds, and segments

May 22, 2025	TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma	L15.86
--------------	---	--------

86

(4) DOES PAGING MAKE THE SYSTEM TOO SLOW?

■ Translation

■ Issue #1: Starting location of the page table is needed

- HW Support: Page-table base register
 - stores active process
 - Facilitates translation

Stored in RAM →

Page Table:

VP0 → PF3
VP1 → PF7
VP2 → PF5
VP3 → PF2

■ Issue #2: Each memory address translation for paging requires an extra memory reference

- HW Support: TLBs (Chapter 19)

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.87

87

PAGING MEMORY ACCESS

```
1. // Extract the VPN from the virtual address
2. VPN = (VirtualAddress & VPN_MASK) >> SHIFT
3.
4. // Form the address of the page-table entry (PTE)
5. PTEAddr = PTBR + (VPN * sizeof(PTE))
6.
7. // Fetch the PTE
8. PTE = AccessMemory(PTEAddr)
9.
10. // Check if process can access the page
11. if (PTE.Valid == False)
12.     RaiseException(SEGMENTATION_FAULT)
13. else if (CanAccess(PTE.ProtectBits) == False)
14.     RaiseException(PROTECTION_FAULT)
15. else
16.     // Access is OK: form physical address and fetch it
17.     offset = VirtualAddress & OFFSET_MASK
18.     PhysAddr = (PTE.PFN << PFN_SHIFT) | offset
19.     Register = AccessMemory(PhysAddr)
```

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.88

88

COUNTING MEMORY ACCESSES

■ Example: Use this Array initialization Code

```
int array[1000];  
...  
for (i = 0; i < 1000; i++)  
    array[i] = 0;
```

■ Assembly equivalent:

```
0x1024 movl $0x0, (%edi,%eax,4)  
0x1028 incl %eax  
0x102c cmpl $0x03e8,%eax  
0x1030 jne 0x1024
```

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.89

89

VISUALIZING MEMORY ACCESSES:
FOR THE FIRST 5 LOOP ITERATIONS

■ Locations:

■ Page table

■ Array

■ Code

■ 50 accesses
for 5 loop
iterations

The figure consists of three vertically stacked bar charts sharing a common x-axis labeled 'Memory Access' ranging from 0 to 50. The top chart, 'Page Table(PA)', shows accesses at indices 1, 11, 21, 31, and 41, with values 1024, 1174, 1124, 1074, and 1024 respectively. The middle chart, 'Array(PA)', shows accesses at indices 2, 12, 22, 32, and 42, with values 7132, 7282, 7232, 7132, and 7282 respectively. The bottom chart, 'Code(PA)', shows accesses at indices 3, 13, 23, 33, and 43, with values 4096, 4146, 4196, 4096, and 4146 respectively. Arrows point from 'Page Table[39]' and 'Page Table[1]' to the first and last bars of the top chart.

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.90

90

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

91

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 22, 2025 TCSS422: Operating Systems [Spring 2025] L15.2

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

92

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

93

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 22, 2025

TCSS422: Operating Systems [Spring 2025]

L15.4

4

94

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

95

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 22, 2025	TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma	L15.96
--------------	---	--------

96

OBJECTIVES – 5/22


- Questions from 5/20
- Assignment 2 - June 5 AOE
- Quiz 3 – Synchronized Array - May 26 AOE
- Tutorial 2 – Pthread/locks/conditions tutorial 5/29 AOE
- Assignment 3 (as a Tutorial) to be posted...
- Chapter 16: Segmentation
- Chapter 17: Free Space Management
- 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 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.97

97



CHAPTER 19:
TRANSLATION
LOOKASIDE BUFFER
(TLB)

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.98

98

TRANSLATION LOOKASIDE BUFFER

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

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.99

99

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' ranging from 0 to 50.

- Top Graph (Page Table):** The y-axis is 'Page Table(PA)' ranging from 1024 to 1224. It shows two series: 'Page Table[39]' (blue squares) and 'Page Table[1]' (black squares). 'Page Table[39]' has five accesses at indices 5, 15, 25, 35, and 45. 'Page Table[1]' has 45 accesses at indices 1 through 45.
- Middle Graph (Array):** The y-axis is 'Array(PA)' ranging from 7232 to 7132. It shows one series 'mov' (blue squares) with five accesses at indices 5, 15, 25, 35, and 45.
- Bottom Graph (Code):** The y-axis is 'Code(PA)' ranging from 4096 to 1124. It shows one series with 50 accesses (black squares) at indices 0 through 49. The accesses are clustered in three regions: indices 0-10 (around 1024-1074), indices 15-25 (around 4096-4146), and indices 30-40 (around 4146-4196).

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.100

100

TRANSLATION LOOKASIDE BUFFER (TLB)

Part of the CPU's Memory Management Unit (MMU)

Address translation cache

The diagram illustrates the address translation process. A CPU provides a Logical Address to the MMU's TLB. If there is a TLB Hit, the Physical Address is retrieved directly from the TLB. If there is a TLB Miss, the MMU consults the Page Table (which contains all v to p entries) to find the Physical Address. The Physical Address then points to the corresponding page in Physical Memory (Page 0, Page 1, Page 2, ..., Page n).

Address Translation with MMU

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.101

101

TRANSLATION LOOKASIDE BUFFER (TLB)

Part of the CPU's Memory Management Unit (MMU)

Address translation cache

The diagram is identical to the one on slide 101, showing the flow from CPU Logical Address to MMU TLB, then to either a direct Physical Address (TLB Hit) or the Page Table (TLB Miss), and finally to Physical Memory. A callout box is added over the MMU section.

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

Address Translation with MMU

May 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.102

102

OBJECTIVES – 5/22

- Questions from 5/20
- Assignment 2 - June 5 AOE
- Quiz 3 – Synchronized Array - May 26 AOE
- Tutorial 2 – Pthread/locks/conditions tutorial 5/29 AOE
- Assignment 3 (as a Tutorial) to be posted...
- Chapter 16: Segmentation
- Chapter 17: Free Space Management
- 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 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.103

103

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 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.104

104

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 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.105

105

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 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.106

106

OBJECTIVES – 5/22

- Questions from 5/20
- Assignment 2 - June 5 AOE
- Quiz 3 – Synchronized Array - May 26 AOE
- Tutorial 2 – Pthread/locks/conditions tutorial 5/29 AOE
- Assignment 3 (as a Tutorial) to be posted...
- Chapter 16: Segmentation
- Chapter 17: Free Space Management
- 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 22, 2025

TCCS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.107

107

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 22, 2025

TCCS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.108

108

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?

	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 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.109

109

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)

	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 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.110

110

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 22, 2025

TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - Tacoma

L15.111

111

QUESTIONS

112