

TCSS 422: OPERATING SYSTEMS

Paging, Translation Lookaside Buffer, and Smaller Tables



Wes J. Lloyd
Institute of Technology
University of Washington - Tacoma

February 28, 2018

TCSS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

OBJECTIVES

- Quiz 4 – Active Reading Chapter 19
- Homework 2 Questions
- Homework 3 Questions
- Ch. 18
 - Introduction to Paging
- Ch. 19
 - Translation Lookaside Buffer (TLB)
- Ch. 20
 - Smaller Tables

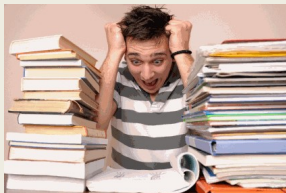
February 28, 2018

TCSS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.2

FEEDBACK FROM 2/23

- There was no feedback !!!



February 28, 2018

TCSS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.3

CHAPTER 18: INTRODUCTION TO PAGING



February 28, 2018

TCSS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.4

PAGING DESIGN QUESTIONS

- Where are page tables stored?
- What are the typical contents of the page table?
- How big are page tables?
- Does paging make the system too slow?

February 28, 2018

TCSS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.5

WHERE ARE PAGE TABLES STORED?

- Example:
 - Consider a 32-bit process address space (up to 4GB)
 - With 4 KB pages
 - 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

February 28, 2018

TCSS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.6

PAGE TABLE EXAMPLE

- With 2^{20} slots in our page table for a single process
- Each slot dereferences a VPN
- Provides physical frame number
- Each slot 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 to store page table for 1 process?
 - 4,194,304 bytes (or 4MB) to index one process

VPN ₀
VPN ₁
VPN ₂
...
...
VPN ₁₀₄₈₅₇₆

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.7

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?

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.8

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
```

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.9

VISUALIZING MEMORY ACCESSES:
FOR THE FIRST 5 LOOP ITERATIONS

- Locations:
 - Page table
 - Array
 - Code
- 50 accesses for 5 loop iterations

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.10

CHAPTER 19:
TRANSLATION
LOOKASIDE BUFFER
(TLB)

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.11

OBJECTIVES

- Chapter 19
 - TLB Algorithm
 - TLB Tradeoffs
 - TLB Context Switch

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.12

TRANSLATION LOOKASIDE BUFFER

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

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.13

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

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.14

TRANSLATION LOOKASIDE BUFFER (TLB)

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

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.15

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

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.16

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

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.17

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)

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.18

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

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.19

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

	00	04	08	12	16
VPN - 00					
VPN - 01					
VPN - 02					
VPN - 03					
VPN - 04					
VPN - 05					
VPN - 06					
VPN - 07					
VPN - 08					
VPN - 09					
VPN - 10					
VPN - 11					
VPN - 12					
VPN - 13					
VPN - 14					
VPN - 15					

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.20

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 - 02					
VPN - 03					
VPN - 04					
VPN - 05					
VPN - 06					
VPN - 07					
VPN - 08					
VPN - 09					
VPN - 10					
VPN - 11					
VPN - 12					
VPN - 13					
VPN - 14					
VPN - 15					

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.21

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 - 02					
VPN - 03					
VPN - 04					
VPN - 05					
VPN - 06					
VPN - 07					
VPN - 08					
VPN - 09					
VPN - 10					
VPN - 11					
VPN - 12					
VPN - 13					
VPN - 14					
VPN - 15					

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.22

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 locality
 - Temporal locality

	00	04	08	12	16
VPN - 00					
VPN - 01					
VPN - 02					
VPN - 03					
VPN - 04					
VPN - 05					
VPN - 06					
VPN - 07					
VPN - 08					
VPN - 09					
VPN - 10					
VPN - 11					
VPN - 12					
VPN - 13					
VPN - 14					
VPN - 15					

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.23

TLB TRADEOFFS

- Page size
 - Larger page sizes increase the probability of a TLB hit
 - Example: 16-bytes (very small), 4096-bytes (common)
- Larger sizes increase memory requirement of offset

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.24

TLB TRADEOFFS - 2

- **Spatial locality**
 - Accessing addresses local to each other improves the hit rate.
 - Consider random vs. sequential array access
- What happens when the data size exceeds the TLB size?
 - E.g. 1st level TLB caches 64 4KB page addresses
 - Single program can cache data lookups for 256 KB

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.25

TLB TRADEOFFS - 3

- **Temporal locality**
 - Higher cache hit ratios are expected for repeated memory accesses close in time
 - Can dramatically improve performance for “second iteration”

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.26

EXAMPLE: LARGE ARRAY ACCESS

- Example: Consider an array of a custom struct where each struct is 64-bytes. Consider sequential access for an array of 8,192 elements stored contiguously in memory:
- 64 structs per 4KB page
- 128 total pages
- TLB caches stores a maximum of 64 - 4KB page lookups
- How many hits vs. misses for sequential array iteration?
 - 1 miss for every 64 array accesses, 63 hits
 - Complete traversal: 128 total misses, 8,064 hits (98.4% hit ratio)

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.27

TLB EXAMPLE IMPLEMENTATIONS

- Intel Nehalem microarchitecture 2008 – multi level TLBs
 - First level TLB: separate cache for data (DTLB) and code (ITLB)
 - Second level TLB: shared TLB (STLB) for data and code
 - Multiple page sizes (4KB, 2MB)
 - Page Size Extension (PSE) CPU flag for larger page sizes
- Intel Haswell microarchitecture 22nm 2013
 - Two level TLB
 - Three page sizes (4KB, 2MB, 1GB)
- Without large page sizes consider the # of TLB entries to address 1.9 MB of memory...

		Page Size	
Name	Level	4 KB	2 MB
DTLB	1st	64	32
ITLB	1st	128	7 / logical core
STLB	2nd	512	none

		Page size		
Name	Level	4 KB	2 MB	1 GB
DTLB	1st	64	32	4
ITLB	1st	128	8 / logical core	none
STLB	2nd	1024	none	none

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.28


HW CACHE TRADEOFF

- **Speed vs. size**

Speed

Size

- In order to be fast, caches must be small
- Too large of a cache will mimic physical memory
- Limitations for on chip memory



“YOU CAN’T HAVE YOUR CAKE AND EAT IT TOO”
FALSE. YOU CAN HAVE A CAKE AND EAT IT. YOU JUST WON’T HAVE IT ANYMORE.

Dwight on “tradeoffs”

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.29

HANDLING TLB MISS

- Historical view
- CISC – Complex instruction set computer
 - Intel x86 CPUs
- Traditionally have provided on CPU HW instructions and handling of TLB misses
- HW has a page table register to store location of page table

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.30

HANDLING TLB MISS - 2

- RISC – Reduced instruction set computer
 - ARM CPUs
 - Traditionally the OS handles TLB misses
 - HW raises an exception
 - Trap handler is executed to handle the miss
- Advantages
 - HW Simplicity: simply needs to raise an exception
 - Flexibility: OS provided page table implementations can use different data structures, etc.

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.31

TLB CONTENTS

- TLB typically may have 32, 64, or 128 entries
- HW searches the entire TLB in parallel to find the translation
- Other bits
 - Valid bit: valid translation?
 - Protection bit: read/execute, read/write
 - Address-space identifier: identify entries by process
 - Dirty bit

VPN	PFN	other bits
-----	-----	------------

Typical TLB entry look like this

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.32

TLB: ON CONTEXT SWITCH

- TLB stores address translations for current running process
- A context/switch to a new process invalidates the TLB
- Must “switch” out the TLB
- **TLB flush**
 - Flush TLB on context switches, set all entries to 0
 - Requires time to flush
 - TLB must be reloaded for each C/S
 - If process not in CPU for long, the TLB may not get reloaded
- **Alternative: be lazy...**
 - Don't flush TLB on C/S
 - Share TLB across processes during C/S
 - Use address space identifier (ASID) to tag TLB entries by process

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.33

TLB: CONTEXT SWITCH - 2

- Address space identifier (ASID): enables TLB data to persist during context switches – also can support virtual machines

VPN	PFN	valid	prot	ASID
10	100	1	rwX	1
-	-	-	-	-
10	170	1	rwX	2
-	-	-	-	-

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.34

SHARED MEMORY SPACE

VPN	PFN	valid	prot	ASID
10	101	1	rwX	1
-	-	-	-	-
50	101	1	rwX	2
-	-	-	-	-

- When processes share a code page
 - Shared libraries ok
 - Code pages typically are RX, not RWX

Sharing of pages is useful as it reduces the number of physical pages in use.

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.35

CACHE REPLACEMENT POLICIES

- When TLB cache is full, how add a new address translation to the TLB?
- Observe how the TLB is loaded / unloaded...
- Goal minimize miss rate, increase hit rate
- **Least Recently Used (LRU)**
 - Evict the oldest entry
- **Random policy**
 - Pick a candidate at random to free-up space in the TLB

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.36

LEAST RECENTLY USED

Reference Row

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1

Page Frame:

7	7	2	2	4	4	0	1	1
0	0	0	0	0	0	3	3	3
1	1	1	1	3	2	2	2	2

- RED – miss
- WHITE – hit
- For 3-page TLB, observe replacement


11 TLB miss, 5 TLB hit

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.37

CHAPTER 20:
PAGING:
SMALLER TABLES



February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.38

OBJECTIVES

- Chapter 20
 - Smaller tables
 - Hybrid tables
 - Multi-level page tables

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.39

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

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.40

LINEAR PAGE TABLES - 2

- Page tables stored in RAM
- Support potential storage of 2^{20} translations
 - = 1,048,576 pages per process @ 4 bytes/page
- Page table size 4MB / process

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

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

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.41

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

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.42

PAGING: USE LARGER PAGES

- **Larger pages** = 16KB = 2^{14}
- 32-bit address space: 2^{32}
- 2^{18} = 262,144 pages

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

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

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.43

PAGE TABLES: WASTED SPACE

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

Page Table

Virtual Address Space

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

Physical Memory

Allocate

PFN	valid	prot	present	dirty
10	1	rw-	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

A 16KB Address Space with 1KB Pages

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.44

PAGE TABLES: WASTED SPACE

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

Page Table

Virtual Address Space

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

Physical Memory

Allocate

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

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

A Page Table For 16KB Address Space

A 16KB Address Space with 1KB Pages


February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.45

HYBRID TABLES

- Combine segments and page tables
- Use stack, heap, code segment base/bound registers
- Base register: point to page table
- Bounds register: store end of page table



February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.46

HYBRID TABLES - 2

- Each process has (3) page tables
- 1 each for code, stack, heap segments
- Base register stores address of start of table
- 2^{16} bits for VPN, can only address 65,536 pages/segment

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

Seg VPN Offset

32-bit Virtual address space with 4KB pages

Seg value	Content
00	unused segment
01	code
10	heap
11	stack

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.47

HYBRID TABLES: COMPUTING PAGE TABLE ADDRESS

- HW must look up page table ADDR on TLB miss
- Segment (SN) bits: indicate which base/bound registers to use

01: SN = (VirtualAddress & SEG_MASK) >> SN_SHIFT

02: VPN = (VirtualAddress & VPN_MASK) >> VPN_SHIFT

03: AddressOfPTE = Base[SN] + (VPN * sizeof(PTE))

- SEG_MASK = 1100 0000 0000 0000 0000 0000 0000 0000
- SN_SHIFT = 30 bits (shift 30 bits right)
 - The SN will just be 2 bits...
- VPN_MASK = 0011 1111 1111 1111 1111 0000 0000 0000
- VPN_SHIFT = 12 bits (shift 12 bits right)
 - The VPN will just be 18 bits...
- PTE ADDR = Base of table + VPN * size of a page table entry
 - PTE=4 (or "10" in binary), will shift VPN 2 bits left ←

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.48

HYBRID TABLE EXAMPLE:

- Consider 3 Segments, w/ 4KB pages
 - 3 code pgs (3 x 4KB), 1 stack pg (1 x 4KB), 3 heap pgs (3 x 4KB)
- 3 sets of base/bounds registers (3 x 16 B)
- 32-bit VPN bit-string:
 - 2 bits – segment type bit code
 - 2 bits – status bits
 - 16 bits – virtual page number VPN (indexes 65,536 pages)
 - 12 bits – page offset (indexes 4KB pages)
- How much memory is required?
 - 4 bytes per PTE x 65,536 pages = 262,144 bytes per segment
 - 3 segments = 786,432 bytes (pg tables) + 48 bytes (registers)
 - 786480 bytes ÷ 1024 KB/byte = ~ 768 KB per process
- How much memory can be addressed?
 - 256 MB (2¹⁶ pages x 4KB)
 - Overhead= 768 KB / 256 MB (.3%)

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.49

HYBRID TABLE EXAMPLE:

- Consider 3 Segments, w/ 4KB pages
 - 3 code pgs (3 x 4KB), 1 stack pg (1 x 4KB), 3 heap pgs (3 x 4KB)
- 3 sets of base/bounds registers (3 x 16 B)
- 32-bit VPN bit-string:
 - 2 bits – segment type bit code
 - 2 bits – status bits
 - 16 bits – virtual page number VPN (indexes 65,536 pages)
 - 12 bits – page offset (indexes 4KB pages)
- How much memory is required?
 - 4 bytes per PTE x 65,536 pages = 262,144 bytes per segment
 - 3 segments = 786,432 bytes (pg tables) + 48 bytes (registers)
 - 786480 bytes ÷ 1024 KB/byte = ~ 768 KB per process
- How much memory can be addressed?
 - 256 MB (2¹⁶ pages x 4KB)
 - Overhead= 768 KB / 256 MB (.3%)

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.50

Problem: For a hybrid approach, with 32-bit VPNs, how do we index *all* RAM for a modern 4GB computer?

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:

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

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.51

MULTI-LEVEL PAGE TABLES - 2

- Add level of indirection, the “page directory”

Linear Page Table

Multi-level Page Table

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

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.52

MULTI-LEVEL PAGE TABLES - 2

- Add level of indirection, the “page directory”

Linear Page Table

Multi-level Page Table

Two level page table:
2²⁰ pages addressed with
two level-indexing
(page directory index, page table index)

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

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.53

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

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

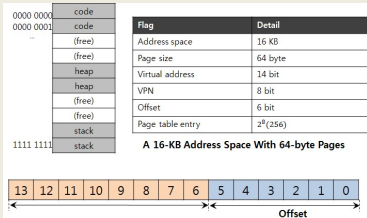
L14.54

Slides by Wes J. Lloyd

L14.9

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)



February 28, 2018

TCS5422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.55

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!**

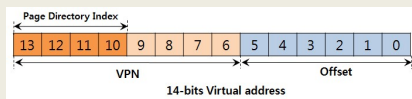
February 28, 2018

TCS5422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.56

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



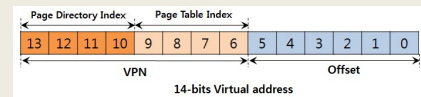
February 28, 2018

TCS5422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.57

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

February 28, 2018

TCS5422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.58

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 !!!

February 28, 2018

TCS5422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.59

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

February 28, 2018

TCS5422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.60

MORE THAN TWO LEVELS

- Consider: page size is $2^9 = 512$ bytes
- Page size 512 bytes / Page entry size 4 bytes
- VPN is 21 bits

Flag	Detail
Virtual address	30 bit
Page size	512 byte
VPN	21 bit
Offset	9 bit

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.61

MORE THAN TWO LEVELS - 2

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

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

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.62

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

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

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.63

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)

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

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.64

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

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

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.65

MORE THAN TWO LEVELS - 4

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

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


- 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
- Savings = $1,536 / 8,388,608$ (8mb) = .0183% !!!

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.66

QUESTIONS



EXAMPLE TLB ENTRY – MIPS R4000

■ Early 64-bit RISC processor

All 64 bits of this TLB entry(example of MIPS R4000)

0 1 2 3 4 5 6 7 8 9 10 11 ... 19 ... 31

VPN												g								ASID			
PFN												C								D V			

Flag	Content
19-bit VPN	The rest reserved for the kernel.
24-bit PFN	Systems can support with up to 64GB of main memory($2^{24} \times 4KB$ pages).
Global bit(G)	Used for pages that are globally-shared among processes.
ASID	OS can use to distinguish between address spaces.
Coherence bit(C)	determine how a page is cached by the hardware.
Dirty bit(D)	marking when the page has been written.
Valid bit(V)	tells the hardware if there is a valid translation present in the entry.

February 28, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L14.68