

UNIVERSITY of WISCONSIN-MADISON  
Computer Sciences Department

CS 537  
Introduction to Operating Systems

Andrea C. Arpaci-Dusseau  
Remzi H. Arpaci-Dusseau

# VIRTUALIZING MEMORY: SMALLER PAGE TABLES

## Questions answered in this lecture:

Review: What are problems with paging?

Review: How large can page tables be?

How can large page tables be avoided with different techniques?

Inverted page tables, segmentation + paging, multilevel page tables

What happens on a TLB miss?

## ANNOUNCEMENTS

- P1: Will be graded by weekend (should be no surprises)
- Project 2: Available now
  - Due Monday, Oct 5
  - **Shell:**
    - What should go in history? (Yes: cmds with errors; No: shell-level errors)
    - Understand child process address space...
- Exam 1: Next week: Thu 10/1 7:15 – 9:15 in Humanities 3650
  - Class time that day for review
  - Look at homeworks / simulations for sample questions
  - **Alternate exam: notify you tomorrow about time**
- Reading for today: Chapter 20

## DISADVANTAGES OF PAGING

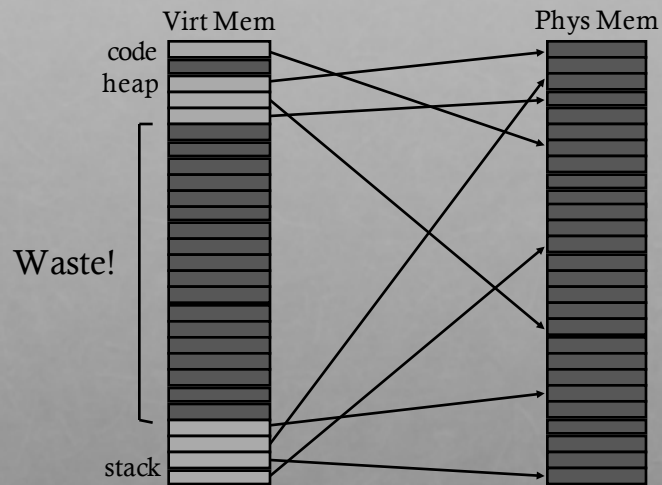
1. Additional memory reference to look up in page table
  - Very inefficient
  - Page table must be stored in memory
  - MMU stores only base address of page table
  - **Avoid extra memory reference for lookup with TLBs (previous lecture)**
2. Storage for page tables may be substantial
  - Simple page table: Requires PTE for all pages in address space
    - Entry needed even if page not allocated
  - Problematic with dynamic stack and heap within address space (today)

## QUIZ: HOW BIG ARE PAGE TABLES?

1. PTE's are **2 bytes**, and **32** possible virtual page numbers  
 $32 * 2 \text{ bytes} = 64 \text{ bytes}$
2. PTE's are **2 bytes**, virtual addrs are **24 bits**, pages are **16 bytes**  
 $2 \text{ bytes} * 2^{(24 - \lg 16)} = 2^{21} \text{ bytes (2 MB)}$
3. PTE's are **4 bytes**, virtual addrs are **32 bits**, and pages are **4 KB**  
 $4 \text{ bytes} * 2^{(32 - \lg 4K)} = 2^{22} \text{ bytes (2 MB)}$
4. PTE's are **4 bytes**, virtual addrs are **64 bits**, and pages are **4 KB**  
 $4 \text{ bytes} * 2^{(64 - \lg 4K)} = 2^{54} \text{ bytes}$

How big is each page table?

# WHY ARE PAGE TABLES SO LARGE?



## MANY INVALID PT ENTRIES

Format of linear page tables:

PFN	valid	prot
10	1	r-x
-	0	-
23	1	rw-
-	0	-
-	0	-
-	0	-
-	0	-
...	many more invalid...	
-	0	-
-	0	-
-	0	-
-	0	-
28	1	rw-
4	1	rw-

how to avoid  
storing these?

## AVOID SIMPLE LINEAR PAGE TABLE

Use more complex page tables, instead of just big array

Any data structure is possible with software-managed TLB

- Hardware looks for vpn in TLB on every memory access
- If TLB does not contain vpn, TLB miss
  - Trap into OS and let OS find vpn->ppn translation
  - OS notifies TLB of vpn->ppn for future accesses

## APPROACH 1: INVERTED PAGE TABLE

Inverted Page Tables

- Only need entries for virtual pages w/ valid physical mappings

Naïve approach:

Search through data structure  $\langle \text{ppn}, \text{vpn} + \text{asid} \rangle$  to find match

- Too much time to search entire table

Better: Find possible matches entries by hashing  $\text{vpn} + \text{asid}$

- Smaller number of entries to search for exact match

Managing inverted page table requires software-controlled TLB

For hardware-controlled TLB, need well-defined, simple approach

## OTHER APPROACHES

1. Inverted Pagetables
2. Segmented Pagetables
3. Multi-level Pagetables
  - Page the page tables
  - Page the pagetables of page tables...

## VALID PTES ARE CONTIGUOUS

	PFN	valid	prot	
	10	1	r-x	
	-	0	-	
	23	1	rw-	
	-	0	-	
	-	0	-	
	-	0	-	
	-	0	-	
	...many more invalid...			
	-	0	-	
	-	0	-	
	-	0	-	
	-	0	-	
	28	1	rw-	
	4	1	rw-	

how to avoid  
storing these?

Note "hole" in addr space:  
valids vs. invalids are clustered

How did OS avoid allocating holes in  
phys memory?

Segmentation

# COMBINE PAGING AND SEGMENTATION

Divide address space into segments (code, heap, stack)

- Segments can be variable length

Divide each segment into fixed-sized pages

Logical address divided into three portions

seg # (4 bits)	page number (8 bits)	page offset (12 bits)
-------------------	----------------------	-----------------------

Implementation

- Each segment has a page table
- Each segment track base (physical address) and bounds of **page table** for that segment

# QUIZ: PAGING AND SEGMENTATION

seg # (4 bits)	page number (8 bits)	page offset (12 bits)
-------------------	----------------------	-----------------------

seg	base	bounds	R	W
0	0x002000	0xff	1	0
1	0x000000	0x00	0	0
2	0x001000	0x0f	1	1

0x002070 read: 0x004070  
 0x202016 read: 0x003016  
 0x104c84 read: error  
 0x010424 write: error  
 0x210014 write: error  
 0x203568 read: 0x02a568

...	
0x01f	0x001000
0x011	
0x003	
0x02a	
0x013	
...	
0x00c	0x002000
0x007	
0x004	
0x00b	
0x006	
...	

# ADVANTAGES OF PAGING AND SEGMENTATION

## Advantages of Segments

- Supports sparse address spaces
  - Decreases size of page tables
  - If segment not used, not need for page table

## Advantages of Pages

- No external fragmentation
- Segments can grow without any reshuffling
- Can run process when some pages are swapped to disk (next lecture)

## Advantages of Both

- Increases flexibility of sharing
  - Share either single page or entire segment
  - How?

# DISADVANTAGES OF PAGING AND SEGMENTATION

## Potentially large page tables (for each segment)

- Must allocate each page table contiguously
- More problematic with more address bits
- Page table size?
  - Assume 2 bits for segment, 18 bits for page number, 12 bits for offset

Each page table is:

= Number of entries \* size of each entry  
 = Number of pages \* 4 bytes  
 =  $2^{18} * 4 \text{ bytes} = 2^{20} \text{ bytes} = 1 \text{ MB!!!}$

## OTHER APPROACHES

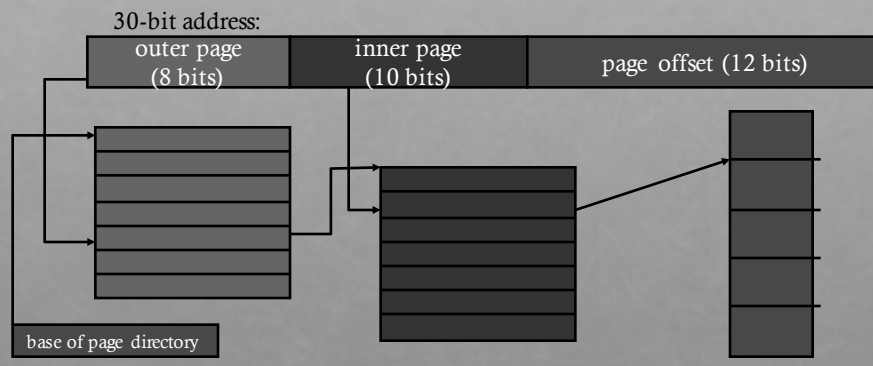
1. Inverted Pagetables
2. Segmented Pagetables
3. Multi-level Pagetables
  - Page the page tables
  - Page the pages of page tables...

## 3) MULTILEVEL PAGE TABLES

Goal: Allow each page tables to be allocated non-contiguously

Idea: Page the page tables

- Creates multiple levels of page tables; outer level “page directory”
- Only allocate page tables for pages in use
- Used in x86 architectures (hardware can walk known structure)





## QUIZ: MULTILEVEL

page directory		page of PT (@PPN:0x3)		page of PT (@PPN:0x92)		
PPN	valid	PPN	valid	PPN	valid	
0x3	1	0x10	1	-	0	
-	0	0x23	1	-	0	
-	0	-	0	-	0	translate 0x01ABC
-	0	-	0	-	0	0x23ABC
-	0	0x80	1	-	0	translate 0x00000
-	0	0x59	1	-	0	0x10000
-	0	-	0	-	0	
-	0	-	0	-	0	
-	0	-	0	-	0	
-	0	-	0	-	0	translate 0xFEED0
-	0	-	0	-	0	0x55ED0
-	0	-	0	-	0	
-	0	-	0	0x55	1	
0x92	1	-	0	0x45	1	

20-bit address:

outer page (4 bits)	inner page (4 bits)	page offset (12 bits)
------------------------	------------------------	-----------------------

## QUIZ: ADDRESS FORMAT FOR MULTILEVEL PAGING

30-bit address:

outer page	inner page	page offset (12 bits)
------------	------------	-----------------------

How should logical address be structured?

- How many bits for each paging level?

Goal?

- Each page table fits within a page
- PTE size \* number PTE = page size
  - Assume PTE size = 4 bytes
  - Page size =  $2^{12}$  bytes = 4KB
  - $2^2$  bytes \* number PTE =  $2^{12}$  bytes
  - $\rightarrow$  number PTE =  $2^{10}$
- $\rightarrow$  # bits for selecting inner page = 10

Remaining bits for outer page:

- $30 - 10 - 12 = 8$  bits

## PROBLEM WITH 2 LEVELS?

Problem: page directories (outer level) may not fit in a page

64-bit address:

outer page?	inner page (10 bits)	page offset (12 bits)
-------------	-------------------------	-----------------------

Solution:

- Split page directories into pieces
- Use another page dir to refer to the page dir pieces.

← VPN →

PD idx 0	PD idx 1	PT idx	OFFSET
----------	----------	--------	--------

How large is virtual address space with 4 KB pages, 4 byte PTEs, each page table fits in page given 1, 2, 3 levels?

4KB / 4 bytes → 1K entries per level

1 level:  $1K * 4K = 2^{22} = 4 \text{ MB}$

2 levels:  $1K * 1K * 4K = 2^{32} \approx 4 \text{ GB}$

3 levels:  $1K * 1K * 1K * 4K = 2^{42} \approx 4 \text{ TB}$

## QUIZ: FULL SYSTEM WITH TLBS

On TLB miss: lookups with more levels more expensive

How much does a miss cost?

Assume 3-level page table

Assume 256-byte pages

Assume 16-bit addresses

Assume ASID of current process is 211

ASID	VPN	PFN	Valid
211	0xbb	0x91	1
211	0xff	0x23	1
122	0x05	0x91	1
211	0x05	0x12	0

How many physical accesses for each instruction? (Ignore previous ops changing TLB)

(a) 0xAA10: movl 0x1111, %edi

0xaa: (TLB miss → 3 for addr trans) + 1 instr fetch

0x11: (TLB miss → 3 for addr trans) + 1 movl

Total: 8

(b) 0xBB13: addl \$0x3, %edi

0xbb: (TLB hit → 0 for addr trans) + 1 instr fetch from 0x9113

Total: 1

(c) 0x0519: movl %edi, 0xFF10

0x05: (TLB miss → 3 for addr trans) + 1 instr fetch

0xff: (TLB hit → 0 for addr trans) + 1 movl into 0x2310

Total: 5

## SUMMARY: BETTER PAGE TABLES

### Problem:

Simple linear page tables require too much contiguous memory

Many options for efficiently organizing page tables

If OS traps on TLB miss, OS can use any data structure

- Inverted page tables (hashing)

If Hardware handles TLB miss, page tables must follow specific format

- Multi-level page tables used in x86 architecture
- Each page table fits within a page

### Next Topic:

What if desired address spaces do not fit in physical memory?

.