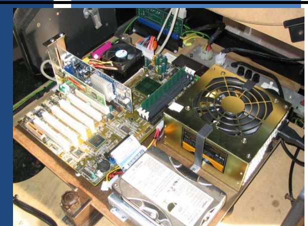
TCSS 422: OPERATING SYSTEMS



Memory Virtualization, Segmentation, Memory Paging

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

November 28, 2018 | TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

FEEDBACK FROM 11/26

- I didn't understand the demo of HW 3. Can you demonstrate it again please?
- Additional Homework 3 Questions
- What is a page table?
- How is address translation performed using page tables?

PAGING MEMORY ACCESS

```
// Extract the VPN from the virtual address
VPN = (VirtualAddress & VPN_MASK) >> SHIFT

// Form the address of the page-table entry (PTE)
PTEAddr = PTBR + (VPN * sizeof(PTE))

// Fetch the PTE
PTE = AccessMemory(PTEAddr)

// Check if process can access the page
if (PTE.Valid == False)
        RaiseException(SEGMENTATION_FAULT)
else if (CanAccess(PTE.ProtectBits) == False)
        RaiseException(PROTECTION_FAULT)
else
        // Access is OK: form physical address and fetch it
        offset = VirtualAddress & OFFSET_MASK
        PhysAddr = (PTE.PFN << PFN_SHIFT) | offset
        Register = AccessMemory(PhysAddr)
```


FEEDBACK - 2

- What is the difference between single-level page tables and multi-level page tables?

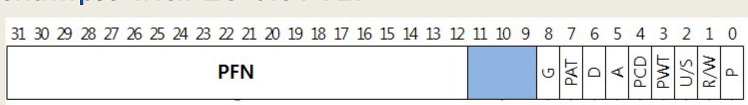
- More examples for paging would be helpful, going step-by-step
- Examples went too fast, especially bits, bytes, etc.
- Multi-level page tables: determining required memory space for tables given:
  - Physical memory size ("the computer")
  - Virtual memory size ("process address space")

PAGE TRANSLATION EXAMPLE

- **Example:**
- Consider a 1-GB computer with 512-byte pages
- Consider a simple hello world program
  - Program has only 4 memory pages
  - 1 code page, 1 stack page, 1 heap page, 1 data segment page
- (1) How many 512-byte memory pages can the computer hold?
- (VPN) The operating system provides each user program a 1GB virtual address space.
- (2) How many VPN bits are required to index any virtual page?

EXAMPLE - 2

- (3) To reference any individual byte on a 512-byte page, how many bits are required (OFFSET bits)?
- A single-level page table provides a one-dimensional array to look up the physical frame number of each virtual memory page
- Each page table entry (PTE) is like a record. It contains the Physical Frame Number (PFN) and status bits for the page
- PTE example with 20-bit PTE:



An x86 Page Table Entry(PTE)

EXAMPLE - 3

- Now consider our Page Table Entry (PTE) for our 1GB computer
- (4) How bits are required for the PFN in the PTE?
- (5) How much capacity (in bits) is available for status bits given the size of our PFN from #4, if we assume our PTE size is 4 bytes?
- (6) What is the storage requirement for a 1-level page table?
- (7) Using 1-level page tables to index memory, how many process would fill main memory with page tables!!??

OBJECTIVES

- Quiz 5
- Program 3
- <u>Paging</u>
- Chapter 20 – Paging Smaller Tables
- Chapter 21/22 – Beyond Physical Memory

CHAPTER 20: PAGING: SMALLER TABLES

OBJECTIVES

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

PAGING: USE LARGER PAGES

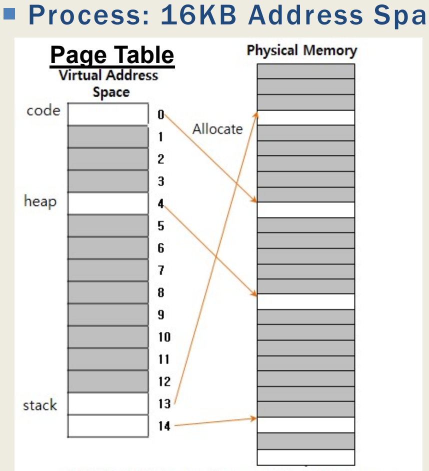
- Larger pages = 16KB = 2^{14}
- 32-bit address space: 2^{32}
- 2^{18} = 262,144 pages

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

- Memory requirement cut to ¼
- However pages are huge
- Internal fragmentation results
- 16KB page(s) allocated for small programs with only a few variables

PAGE TABLES: WASTED SPACE

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



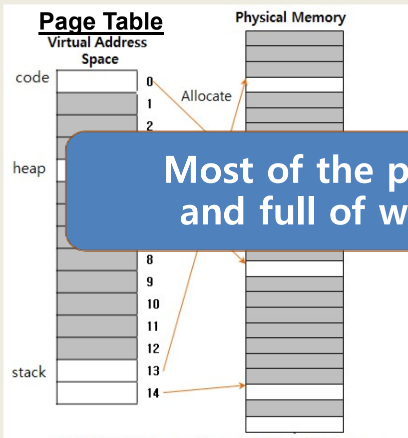
A 16KB Address Space with 1KB Pages

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

A Page Table For 16KB Address Space

PAGE TABLES: WASTED SPACE

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



A 16KB Address Space with 1KB Pages

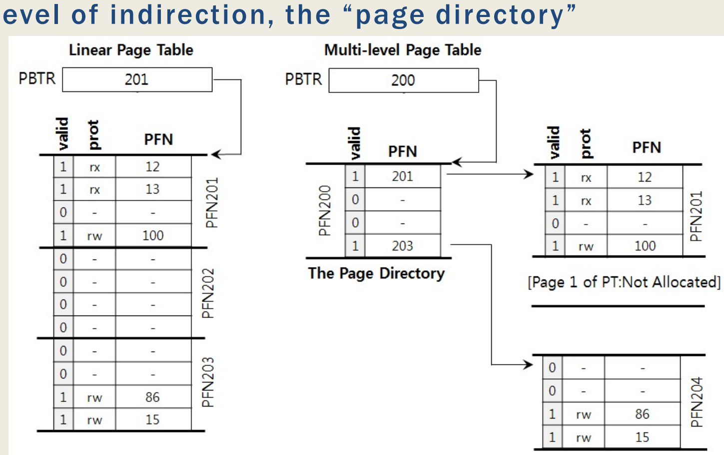
Most of the page table is unused and full of wasted space. (73%)

PFN	valid	prot	present	dirty
[illegible]	[illegible]	[illegible]	[illegible]	0
[illegible]	[illegible]	[illegible]	[illegible]	-
[illegible]	[illegible]	[illegible]	[illegible]	-
[illegible]	[illegible]	[illegible]	[illegible]	-
15	1	rw-	1	1
…	…	…	…	…
-	0	-	-	-
3	1	rw-	1	1
23	1	rw-	1	1

A Page Table For 16KB Address Space

MULTI-LEVEL PAGE TABLES - 2

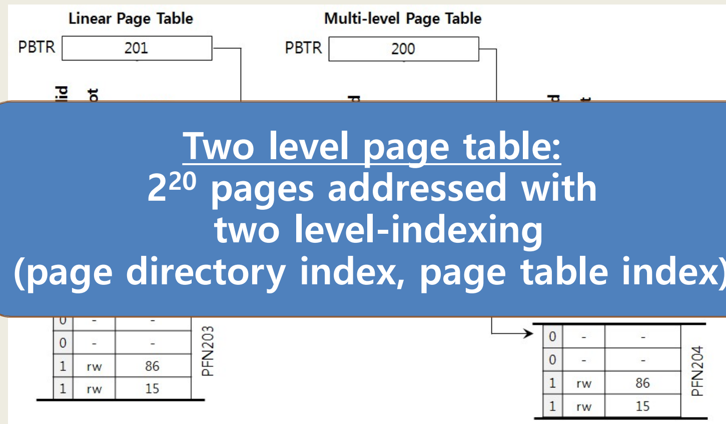
- Add level of indirection, the “page directory”



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

MULTI-LEVEL PAGE TABLES - 2

- Add level of indirection, the "page directory"



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

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

November 28, 2018 | TCSS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma | L17.15

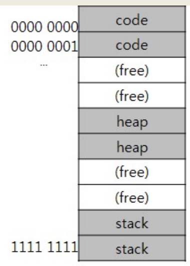
MULTI-LEVEL PAGE TABLES - 3

- 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

November 28, 2018 | TCSS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma | L17.16

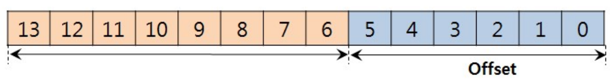
1-LEVEL PAGING EXAMPLE

- Given a 16KB address space, with 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)



Flag	Detail
Address space	16 KB
Page size	64 byte
Virtual address	14 bit
VPN	8 bit
Offset	6 bit
Page table entry	2^8(256)

A 16-KB Address Space With 64-byte Pages



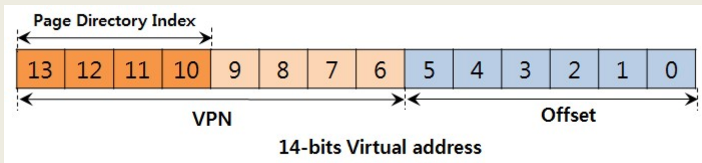
1-LEVEL PAGING EXAMPLE - 2

- If paging requires 256 total page table entries (4 bytes each)
- Yields 1,024 bytes page table size
- Page table stored using 64-byte pages
- (1024/64) = 16 memory pages to store page table

- Key idea: the page table is stored using pages too!

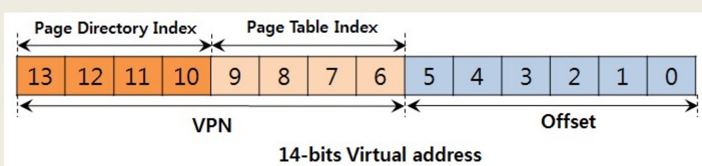
PAGE DIRECTORY INDEX

- 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



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

2-LEVEL EXAMPLE

- For this example, how much space is required to store as a ***single-level*** page table with any number of PTEs?
- Fully populated address space (all_memory.c)
- The full memory space is mapped:
- 16 page directory entries (PDE) x 16 page table entries (PTE) = 256 total PTEs
- (1) How much memory is required for the PD?
- (2) How much memory is required for the PT?
- (3) What is the total memory required to map all_memory.c ?
- 16KB address space, 64 byte pages
- 256 page frames, 4 byte page size
- 1,024 bytes required (*single level*)

2-LEVEL EXAMPLE – 2

- How much space is required for a ***two-level*** page table with only 4 page table entries (PTEs) ?
- Nearly empty address space
- hello.c – 4 total pages: stack, heap, code, data:
- Page directory = 16 entries x 4 bytes (1 x 64 byte page)
- Page table = 4 entries x 4 bytes (1 x 64 byte page)
- (4) How much memory is required for the PDs?
- (5) How much memory is required for the PTs?
- (6) What is the total memory required to map hello.c ?
- 128 bytes required (2 x 64 byte pages)
  - Savings = using just 12.5% the space !!! (128/1024 single-level)

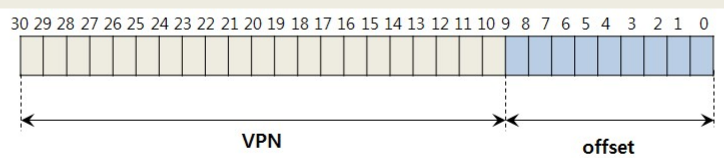
32-BIT EXAMPLE

- 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
- Memory savings = using just .78 % the space !!! (8KB/4MB)
- 100 sparse processes now require < 1MB for page tables

November 28, 2018	TCSS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L17.23

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

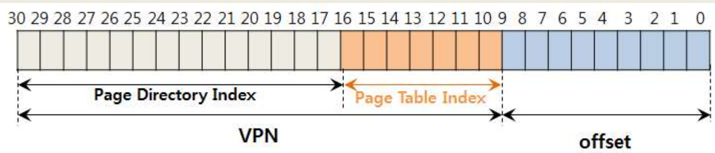


Flag	Detail
Virtual address	30 bit
Page size	512 byte
VPN	21 bit
Offset	9 bit

November 28, 2018	TCSS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L17.24

MORE THAN TWO LEVELS - 2

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

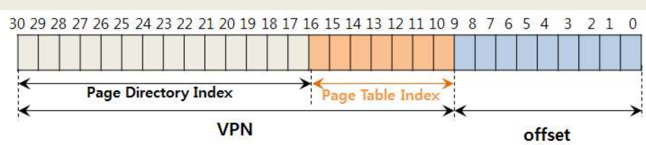


Flag	Detail
Virtual address	30 bit
Page size	512 byte
VPN	21 bit
Offset	9 bit
Page entry per page	128 PTEs

→ $\log_2 128 = 7$

MORE THAN TWO LEVELS - 3

- To map 1 GB address space (2^{30}=1GB 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

→ $\log_2 128 = 7$

MORE THAN TWO LEVELS - 3

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

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

Virtual address	30 bit
Page size	512 byte
VPN	21 bit
Offset	9 bit
Page entry per page	128 PTEs

→ $\log_2 128 = 7$

MORE THAN TWO LEVELS - 3

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

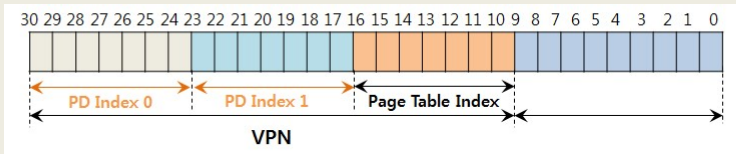
**Need three level page table:
Page directory 0 (PD Index 0)
Page directory 1 (PD Index 1)
Page Table Index**

Virtual address	30 bit
Page size	512 byte
VPN	21 bit
Offset	9 bit
Page entry per page	128 PTEs

→ $\log_2 128 = 7$

MORE THAN TWO LEVELS - 4

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



- Consider the implications for address translation!
- How much space is required for a virtual address space with 4 entries on a 512-byte page? (let’s say 4 32-bit integers)
- PD0 1 page, PD1 1 page, PT 1 page = 1,536 bytes
- Memory Usage= 1,536 (*3-level*) / 8,388,608 (*1-level*) = .0183% !!!

ADDRESS TRANSLATION CODE

```
// 5-level Linux page table address lookup
//
// Inputs:
// mm_struct – process’s memory map struct
// vpage – virtual page address

// Define page struct pointers
pgd_t *pgd;
p4d_t *p4d;
pud_t *pud;
pmd_t *pmt;
pte_t *pte;
struct page *page;
```

ADDRESS TRANSLATION - 2

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

pgd_offset():
Takes a vpage address and the mm_struct for the process, returns the PGD entry that covers the requested address…

p4d/pud/pmd_offset():
Takes a vpage address and the pgd/p4d/pud entry and returns the relevant p4d/pud/pmd.

pte_unmap()
release temporary kernel mapping for the page table entry

INVERTED PAGE TABLES

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

MULTI-LEVEL PAGE TABLE EXAMPLE

- Consider a 16 MB computer which indexes memory using 4KB pages
- (#1) For a single level page table, how many pages are required to index memory?
- (#2) How many bits are required for the VPN?
- (#3) Assuming each page table entry (PTE) can index any byte on a 4KB page, how many offset bits are required?
- (#4) Assuming there are 8 status bits, how many bytes are required for each page table entry?

MULTI LEVEL PAGE TABLE EXAMPLE - 2

- (#5) How many bytes (or KB) are required for a single level page table?
- Let's assume a simple HelloWorld.c program.
- HelloWorld.c requires virtual address translation for 4 pages:
  - 1 – code page   1 – stack page
  - 1 – heap page   1 – data segment page
- (#6) Assuming a two-level page table scheme, how many bits are required for the Page Directory Index (PDI)?
- (#7) How many bits are required for the Page Table Index (PTI)?

MULTI LEVEL PAGE TABLE EXAMPLE - 3

- Assume each page directory entry (PDE) and page table entry (PTE) requires 4 bytes:
  - 6 bits for the Page Directory Index (PDI)
  - 6 bits for the Page Table Index (PTI)
  - 12 offset bits
  - 8 status bits
- (#8) How much **total** memory is required to index the HelloWorld.c program using a two-level page table when we only need to translate 4 total pages?
- HINT: we need to allocate one Page Directory and one Page Table…
- HINT: how many entries are in the PD and PT

MULTI LEVEL PAGE TABLE EXAMPLE - 4

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

ANSWERS

- #1 – 4096 pages
- #2 – 12 bits
- #3 – 12 bits
- #4 – 4 bytes
- #5 – 4096 x 4 = 16,384 bytes (16KB)
- #6 – 6 bits
- #7 – 6 bits
- #8 – 256 bytes for Page Directory (PD) (64 entries x 4 bytes)
  256 bytes for Page Table (PT) <u>TOTAL = 512 bytes</u>
- #9 – 64 entries, where each entry maps a 4,096 byte page
  With 12 offset bits, can address 262,144 bytes (256 KB)
- #10- 512/16384 = .03125 → 3.125%

QUESTIONS