TCSS 422: OPERATING SYSTEMS

Paging
Smaller Tables

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

OBJECTIVES

- Smaller tables
- Hybrid tables
- Multi-level page tables

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

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.

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

PAGING: USE SMALLER TABLES

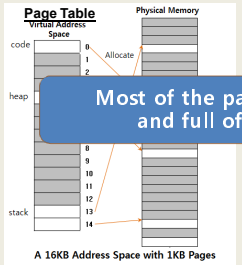
- **Larger pages** = 16KB = 2^{14}
- 32-bit address space: 2^{32}
- 2^{18} = 262,144 pages

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

- 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

Most of the page table is unused and full of wasted space.

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

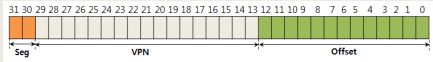
A Page Table For 16KB Address Space

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

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



32-bit Virtual address space with 4KB pages

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

COMPUTING PAGE TABLE ADDESS

- HW must look up page table on TLB miss
- SN bits indicate which pair of 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)
- VPN_MASK = 0011 1111 1111 1111 1111 0000 0000 0000
- VPN_SHIFT = 12 bits (shift 12 bits right)
- PTE ADDR = Base of table + VPN * size of a page table entry

HYBRID TABLES - 3

- Consider memory storage requirements for hybrid page tables
- 3 Segments, w/ 4KB pages
  - 3 code pages
  - 1 stack pages
  - 3 heap pages
- 3 sets of base/bounds registers
- 32-bit bit-string:
  - 2 bits – segment type
  - 16 bits – virtual
  - 12 bits – page offset
- How much memory is required?
- How much memory can be addressed?

HYBRID TABLES - 4

- Consider a large sparsely populated heap
- Heap may have been enlarged for memory which was freed
- Free-ing memory doesn't necessarily shrink the heap
- Consider our realloc example
- Large sparse segments waste space

- Page tables are now of variable size (no longer fixed)
- Can be 1 to many actual pages
- Must find space for variable sized page tables
- Fragmentation is possible
- Need a free space list

MULTI-LEVEL PAGE TABLES

- Consider a page table:
- 32-bit addressing, 4KB pages
- 2^{20} page table entries
- Even if memory is sparsely populated the *per process* page table requires:

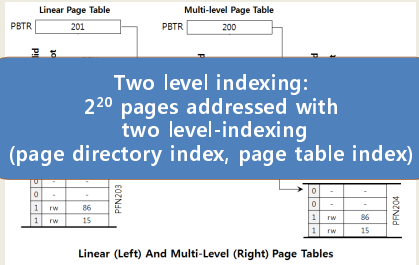
$$\text{Page table size} = \frac{2^{32}}{2^{12}} * 4Byte = 4MByte$$

- 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!

MULTI-LEVEL PAGE TABLES - 2

- Add level of indirection, the "page directory"



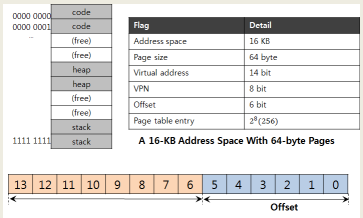
Linear (Left) And Multi-Level (Right) Page Tables

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

EXAMPLE

- 16KB address space, 64byte pages
- How large would a one-level page table need to be?
- $2^{14} / 2^6 = 2^8 = 256$ entries (page frames)



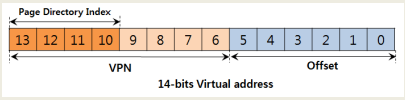
A 16-KB Address Space With 64-byte Pages

EXAMPLE - 2

- 256 total page table entries (4 bytes each)
- 1,024 bytes page table size / stored using 64-byte pages = 16 page directory entries (PDEs)
- Each page directory page can hold 16 page table entries (PTEs)
- 16 page directory pages x 16 page table entries = 256 total page table entries

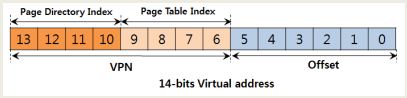
PAGE DIRECTORY INDEX

- 8 bit VPN to map 256 pages
- 4 bits for page directory index
- 6 bits offset into 64-byte page



PAGE TABLE INDEX

- 4 bits page directory index
- 4 bits page table index



- Can now have just one page directory entry
- Plus a 16-entry (64-byte) page

EXAMPLE - 3

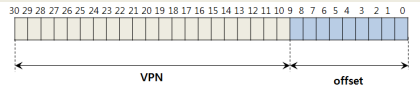
- For this example, how much space is required to store as a single-level array based page table?
- 16KB address space, 64 byte pages
- 256 page frames, 4 byte page size
- 1,024 bytes required (single level)
- How much space is required for a two-level page table with only 4 page table entries?
- Page 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 !!!

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:
- 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

MORE THAN TWO LEVELS

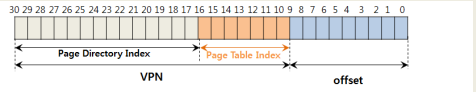
- Consider: page size is 2^9 = 512 bytes
- Page size 512 bytes / Page entry size 4 bytes



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

MORE THAN TWO LEVELS - 2

- Page table entries per page = 512 / 4 = 128
- 7 bites – for 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 → $\log_2 128 = 7$

MORE THAN TWO LEVELS - 3

- To map an address space of 2^{30}=1GB memory locations
- 2^{14} = 16,384 page directory entries are required
- Wit
- Pag

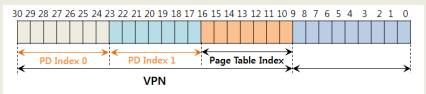
Requires three level page table:
Page directory 0 (PD Index 0)
Page directory 1 (PD Index 1)
Page Table Index

VPN offset

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

MORE THAN TWO LEVELS - 4

- We can now address 1GB using "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
- Savings = 1,536 / 2,097,152 = .07% !!!

ADDRESS TRANSLATION - 1

```
01:     VPN = (VirtualAddress & VPN_MASK) >> SHIFT
02:     (Success,TlbEntry) = TLB_Lookup(VPN)
03:     if(Success == True)       //TLB Hit
04:       if(CanAccess(TlbEntry.ProtectBits) == True)
05:             Offset = VirtualAddress & OFFSET_MASK
06:             PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
07:             Register = AccessMemory(PhysAddr)
08:       else RaiseException(PROTECTION_FAULT);
09:     else // perform the full multi-level lookup
```

(05-07) Generate physical address from TLB

ADDRESS TRANSLATION - 2

```
11:     else
12:             PDIndex = (VPN & PD_MASK) >> PD_SHIFT
13:             PDEAddr = PDBR + (PDIndex * sizeof(PDE))
14:             PDE = AccessMemory(PDEAddr)
15:             if(PDE.Valid == False)
16:                     RaiseException(SEGMENTATION_FAULT)
17:             else // PDE is Valid: now fetch PTE from PT
```

(15-17) Check if PDE is valid, if so fetch entry from page table

ADDRESS TRANSLATION - 3

```
18:     PTIndex = (VPN & PT_MASK) >> PT_SHIFT
19:     PTEAddr = (PDE.PFN << SHIFT) + (PTIndex * sizeof(PTE))
20:     PTE = AccessMemory(PTEAddr)
21:     if(PTE.Valid == False)
22:             RaiseException(SEGMENTATION_FAULT)
23:     else if(CanAccess(PTE.ProtectBits) == False)
24:             RaiseException(PROTECTION_FAULT);
25:     else
26:             TLB_Insert(VPN, PTE.PFN , PTE.ProtectBits)
27:             RetryInstruction()
```


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
- Finding process memory pages requires search of 2^{20} pages
- Hash table: can index memory and speed lookups

QUESTIONS