

Virtual Memory

Some slides based on those provided by the authors of the text

Topics

- Motivations for VM
- Address translation
- Accelerating translation with TLBs

Motivations for Virtual Memory

- **Use Physical DRAM as a Cache for the Disk**
 - Address space of a process can exceed physical memory size
 - Sum of address spaces of multiple processes can exceed physical memory
- **Simplify Memory Management**
 - Multiple processes resident in main memory.
 - Each process with its own address space
 - Only "active" code and data is actually in memory
 - Allocate more memory to process as needed.
- **Provide Protection**
 - One process can't interfere with another.
 - because they operate in different address spaces.
 - User process cannot access privileged information
 - different sections of address spaces have different permissions.

CompOrg Fall - Virtual Memory 2

Motivation #1: DRAM a "Cache" for Disk

Full address space is quite large:

- 32-bit addresses: -4,000,000,000 (4 billion) bytes
- 64-bit addresses: -16,000,000,000,000,000,000 (16 quintillion) bytes

Disk storage is ~156X cheaper than DRAM storage

- 8 GB of DRAM: - \$10,000
- 8 GB of disk: - \$64

To access large amounts of data in a cost-effective manner, the bulk of the data must be stored on disk

```

    graph LR
      SRAM[4 MB: ~$400] <--> DRAM[256 MB: ~$320]
      DRAM <--> Disk[(8 GB: ~$64)]
      style SRAM fill:#ccc,stroke:#333,stroke-width:1px
      style DRAM fill:#ccc,stroke:#333,stroke-width:1px
      style Disk fill:#ccc,stroke:#333,stroke-width:1px
    
```

CompOrg Fall - Virtual Memory 3

Levels in Memory Hierarchy

	Register	Cache	Memory	Disk Memory
size:	32 B	32 KB-4MB	128 MB	30 GB
speed:	3 ns	6 ns	60 ns	8 ms
\$/Mbyte:		\$100/MB	\$1.25/MB	\$0.008/MB
line size:	8 B	32 B	4 KB	

larger, slower, cheaper →

CompOrg Fall - Virtual Memory 4

DRAM vs. SRAM as a "Cache"

DRAM vs. disk is more extreme than SRAM vs. DRAM

- Access latencies:**
 - DRAM ~10X slower than SRAM
 - Disk ~100,000X slower than DRAM
- Importance of exploiting spatial locality:**
 - First byte is ~100,000X slower than successive bytes on disk
 - » vs. ~4X improvement for page-mode vs. regular accesses to DRAM
- Bottom line:**
 - Design decisions made for DRAM caches driven by enormous cost of misses

CompOrg Fall - Virtual Memory 5

Impact of These Properties on Design

If DRAM was to be organized similar to an SRAM cache, how would we set the following design parameters?

- Line size?**
 - Large, since disk better at transferring large blocks
- Associativity?**
 - High, to minimize miss rate
- Write through or write back?**
 - Write back, since can't afford to perform small writes to disk

What would the impact of these choices be on:

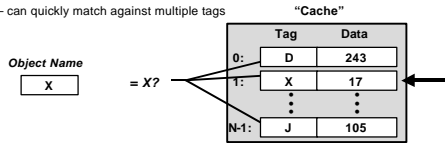
- miss rate**
 - Extremely low. << 1%
- hit time**
 - Must match cache/DRAM performance
- miss latency (penalty)**
 - Very high. ~20ms
- tag storage overhead**
 - Low, relative to block size

CompOrg Fall - Virtual Memory 6

Locating an Object in a "Cache"

SRAM Cache

- Tag stored with cache line
- Maps from cache block to memory blocks
 - From cached to uncached form
- No tag for block not in cache
- Hardware retrieves information
 - can quickly match against multiple tags



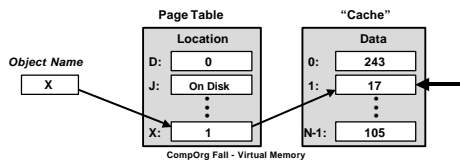
CompOrg Fall - Virtual Memory

7

Locating an Object in a "Cache" (cont.)

DRAM Cache

- Each allocated page of virtual memory has entry in *page table*
 - From uncached form to cached form
- Page table entry even if page not in memory
 - Specifies disk address
- OS retrieves information



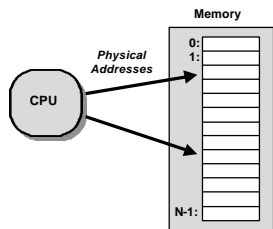
CompOrg Fall - Virtual Memory

8

A System with Physical Memory Only

Examples:

- most Cray machines, early PCs, nearly all embedded systems, etc.



Addresses generated by the CPU point directly to bytes in physical memory

CompOrg Fall - Virtual Memory

9

A System with Virtual Memory

Examples:

- workstations, servers, modern PCs, etc.

Address Translation: Hardware converts *virtual addresses* to *physical addresses* via an OS-managed lookup table (*page table*)

10

Page Faults (Similar to “Cache Misses”)

What if an object is on disk rather than in memory?

- Page table entry indicates virtual address not in memory
- OS exception handler invoked to move data from disk into memory
 - current process suspends, others can resume
 - OS has full control over placement, etc.

Before fault

After fault

CompOrg Fall - Virtual Memory

11

Servicing a Page Fault

Processor Signals Controller

- Read block of length P starting at disk address X and store starting at memory address Y

Read Occurs

- Direct Memory Access (DMA)
- Under control of I/O controller

I/O Controller Signals Completion

- Interrupt the processor
- OS resumes suspended process

CompOrg Fall - Virtual Memory

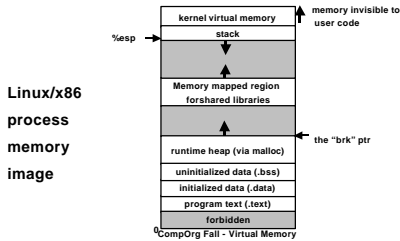
12

Motivation #2: Memory Management

Multiple processes can reside in physical memory.

How do we resolve address conflicts?

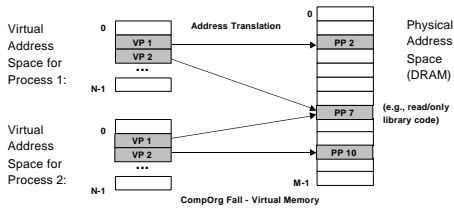
- what if two processes access something at the same address?



13

Solution: Separate Virtual Addr. Spaces

- Virtual and physical address spaces divided into equal-sized blocks
 - blocks are called "pages" (both virtual and physical)
- Each process has its own virtual address space
 - operating system controls how virtual pages as assigned to physical memory
 - every program can start at the same address (virtual address)!

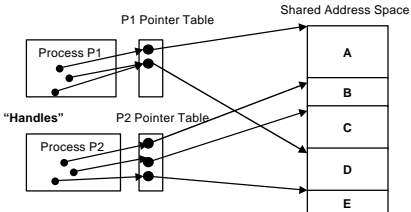


14

Contrast: Macintosh Memory Model

MAC OS 1-9

- Does not use traditional virtual memory



All program objects accessed through "handles"

- Indirect reference through pointer table
- Objects stored in shared global address space

CompOrg Fall - Virtual Memory

15

Macintosh Memory Management

Allocation / Deallocation

- Similar to free-list management of malloc/free

Compaction

- Can move any object and just update the (unique) pointer in pointer table

The diagram illustrates memory management in Macintosh. On the left, two processes, Process P1 and Process P2, are shown. Each process has a pointer table. Process P1's pointer table contains pointers to objects B, A, and C. Process P2's pointer table contains pointers to objects D and E. On the right, a 'Shared Address Space' contains five memory blocks labeled B, A, C, D, and E. Arrows indicate the mapping from the pointer tables to the objects in the shared address space. The text 'Handles' is written near the pointer tables, and 'CompOrg Fall - Virtual Memory' is at the bottom left of the diagram area.

Mac vs. VM-Based Memory Mgmt

Allocating, deallocating, and moving memory:

- can be accomplished by both techniques

Block sizes:

- **Mac: variable-sized**
 - may be very small or very large
- **VM: fixed-size**
 - size is equal to one *page* (4KB on x86 Linux systems)

Allocating contiguous chunks of memory:

- **Mac: contiguous allocation is required**
- **VM: can map contiguous range of virtual addresses to disjoint ranges of physical addresses**

Protection

- **Mac: "wild write" by one process can corrupt another's data**

CompOrg Fall - Virtual Memory 17

MAC OS X

"Modern" Operating System

- Virtual memory with protection
- Preemptive multitasking
 - Other versions of MAC OS require processes to voluntarily relinquish control

Based on MACH OS

- Developed at CMU in late 1980's

CompOrg Fall - Virtual Memory 18

Motivation #3: Protection

Page table entry contains access rights information

- hardware enforces this protection (trap into OS if violation occurs)

Page Tables

	Read?	Write?	Physical Addr
Process i: VP 0:	Yes	No	PP 9
VP 1:	Yes	Yes	PP 4
VP 2:	No	No	XXXXXXXX
⋮	⋮	⋮	⋮

Memory

0:	
1:	
⋮	⋮
N-1:	

CompOrg Fall - Virtual Memory 19

Address Translation

Address translation must be fast (it happens on every memory access).

We need a fully associative placement policy.

- miss penalty is huge!

We can't afford to go looking at every virtual page to find the right one

- we don't use the *tag bits* approach

CompOrg Fall - Virtual Memory 20

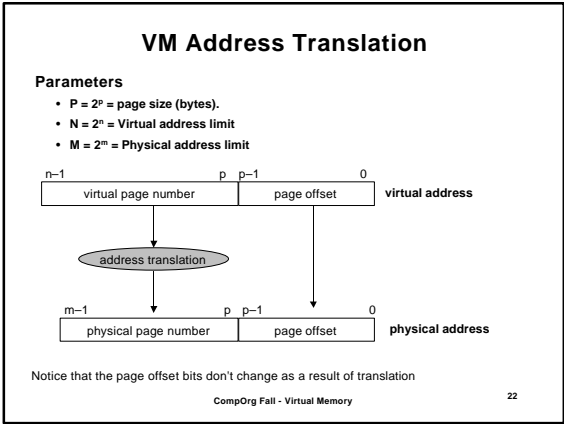
VM Address Translation

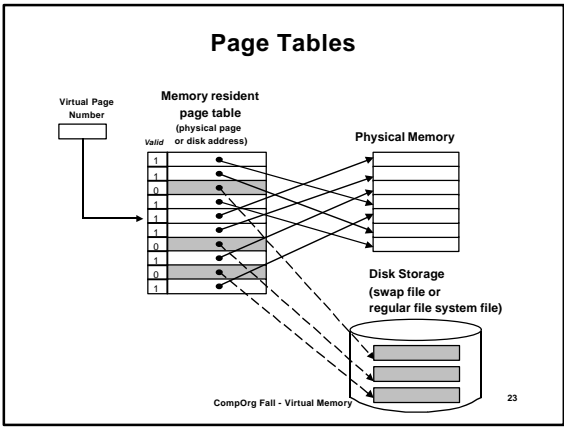
$V = \{0, 1, \dots, N-1\}$ virtual address space
 $P = \{0, 1, \dots, M-1\}$ physical address space $N > M$

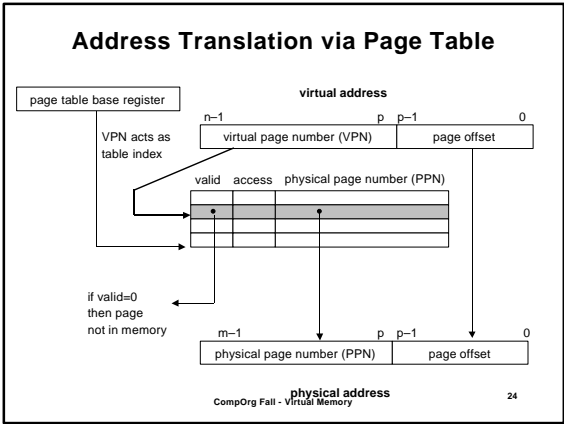
MAP: $V \rightarrow P \cup \{\emptyset\}$ address mapping function

MAP(a) = a' if data at virtual address *a* is present at physical address *a'* in P
 = \emptyset if data at virtual address *a* is not present in P

CompOrg Fall - Virtual Memory 21







Page Table Operation

Translation

- Separate (set of) page table(s) per process
- VPN forms index into page table (points to a page table entry)

Computing Physical Address

- Page Table Entry (PTE) provides information about page
 - if (valid bit = 1) then the page is in memory.
 - » Use physical page number (PPN) to construct address
 - if (valid bit = 0) then the page is on disk
 - » Page fault
 - » Must load page from disk into main memory before continuing

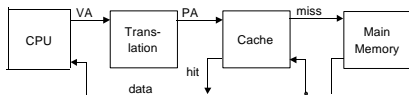
Checking Protection

- Access rights field indicate allowable access
 - e.g., read-only, read-write, execute-only
 - typically support multiple protection modes (e.g., kernel vs. user)
- Protection violation fault if user doesn't have necessary permission

CompOrg Fall - Virtual Memory

25

Integrating VM and Cache



Most Caches "Physically Addressed"

- Accessed by physical addresses
- Allows multiple processes to have blocks in cache at same time
- Allows multiple processes to share pages
- Cache doesn't need to be concerned with protection issues
 - Access rights checked as part of address translation

Perform Address Translation Before Cache Lookup

- But this could involve a memory access itself (of the PTE)
- Of course, page table entries can also become cached

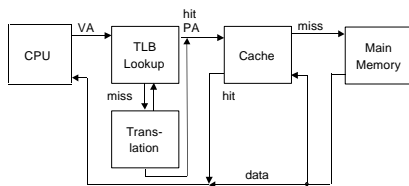
CompOrg Fall - Virtual Memory

26

Speeding up PA Translation with a TLB

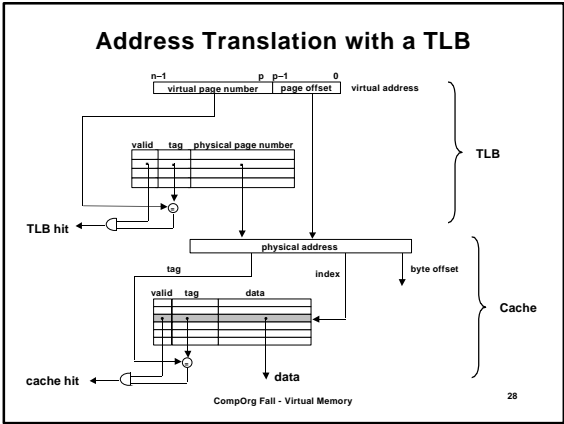
"Translation Lookaside Buffer" (TLB)

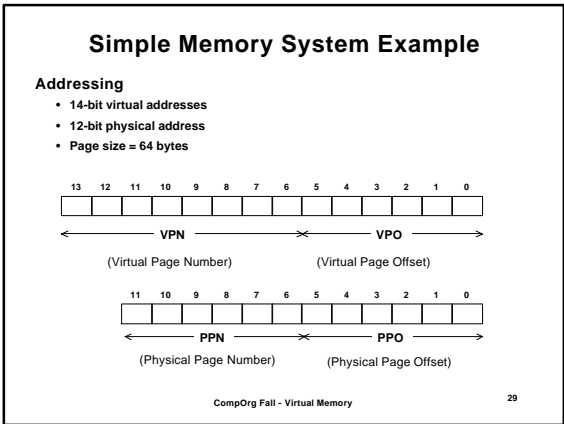
- Small hardware cache in MMU
- Maps virtual page numbers to physical page numbers
- Contains complete page table entries for small number of pages



CompOrg Fall - Virtual Memory

27





Simple Memory System Page Table

- Only show first 16 entries

VPN	PPN	Valid	VPN	PPN	Valid
00	28	1	08	13	1
01	-	0	09	17	1
02	33	1	0A	09	1
03	02	1	0B	-	0
04	-	0	0C	-	0
05	16	1	0D	2D	1
06	-	0	0E	11	1
07	-	0	0F	0D	1

CompOrg Fall - Virtual Memory 30

Multi-Level Page Tables

Given:

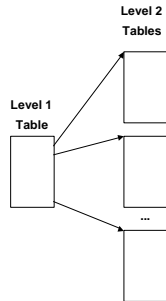
- 4KB (2^{12}) page size
- 32-bit address space
- 4-byte PTE

Problem:

- Would need a 4 MB page table!
 - $2^{20} \cdot 4$ bytes

Common solution

- multi-level page tables
- e.g., 2-level table (P6)
 - Level 1 table: 1024 entries, each of which points to a Level 2 page table.
 - Level 2 table: 1024 entries, each of which points to a page



CompOrg Fall - Virtual Memory

34

Main Themes

Programmer's View

- Large "flat" address space
 - Can allocate large blocks of contiguous addresses
- Processor "owns" machine
 - Has private address space
 - Unaffected by behavior of other processes

System View

- User virtual address space created by mapping to set of pages
 - Need not be contiguous
 - Allocated dynamically
 - Enforce protection during address translation
- OS manages many processes simultaneously
 - Continually switching among processes
 - Especially when one must wait for resource
 - » E.g., disk I/O to handle page fault

CompOrg Fall - Virtual Memory

35
