Virtual Memory

Dr. Steve Goddard

goddard@cse.unl.edu

http://cse.unl.edu/~goddard/Courses/JDEP284

Motivations for Virtual Memory

- Use Physical DRAM as a Cache for the 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 ~300X cheaper than DRAM storage:
  - 80 GB of DRAM: ~$33,000
  - 80 GB of disk: ~ $110
- To access large amounts of data in a cost-effective manner, the bulk of the data must be stored on disk:

<table>
<thead>
<tr>
<th>Size</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 MB</td>
<td>~$500</td>
</tr>
<tr>
<td>10 GB</td>
<td>~$200</td>
</tr>
<tr>
<td>80 GB</td>
<td>~$110</td>
</tr>
</tbody>
</table>

Levels in Memory Hierarchy

<table>
<thead>
<tr>
<th>CPU</th>
<th>Register</th>
<th>Cache</th>
<th>Memory</th>
<th>Disk Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>32 B</td>
<td>32 KB-4 MB</td>
<td>1024 MB</td>
<td>100 GB</td>
</tr>
<tr>
<td>speed</td>
<td>1 ns</td>
<td>2 ns</td>
<td>30 ns</td>
<td>8 ms</td>
</tr>
<tr>
<td>S/MB/s</td>
<td>$125/MB</td>
<td>$0.20/MB</td>
<td>$0.001/MB</td>
<td></td>
</tr>
<tr>
<td>line size</td>
<td>8 B</td>
<td>32 B</td>
<td>4 KB</td>
<td></td>
</tr>
</tbody>
</table>

larger, slower, cheaper
**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

**Impact of 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
  - Very high. ~20ms
- Tag storage overhead
  - Low, relative to block size

**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
- Save a few bits by only storing tag
- No tag for block not in cache
- Hardware retrieves information
  - Can quickly match against multiple tags

**Locating an Object in “Cache” (cont.)**

**DRAM Cache**
- Each allocated page of virtual memory has entry in page table
- Mapping from virtual pages to physical pages
  - From uncached form to cached form
  - Page table entry even if page not in memory
  - Specifies disk address
  - Only way to indicate where to find page
- OS retrieves information

**A System with Physical Memory Only**

Examples:
- most Cray machines, early PCs, nearly all embedded systems, etc.
- Addresses generated by the CPU correspond directly to bytes in physical memory

**A System with Virtual Memory**

Examples:
- workstations, servers, modern PCs, etc.
- Address Translation: Hardware converts virtual addresses to physical addresses via OS-managed lookup table (page table)
Page Faults (like “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 suspended, others can continue
- OS has full control over placement, etc.

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 processor
- OS resumes suspended process

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?

Solution: Separate Virt. 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 are assigned to physical memory

Contrast: Macintosh Memory Model

MAC OS 1–9
- Does not use traditional virtual memory
- P1 Pointer Table
- P2 Pointer Table
- “Handles”

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

Page 3
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

Motivation #3: Protection

Page table entry contains access rights information
- hardware enforces this protection (trap into OS if violation occurs)

VM Address Translation

Virtual Address Space
- \( V = \{0, 1, ..., N-1\} \)

Physical Address Space
- \( P = \{0, 1, ..., M-1\} \)
  - \( M < N \)

Address Translation
- \( MAP: V \rightarrow P \cup \{\emptyset\} \)
  - For virtual address \( a \):
    - \( MAP(a) = a' \) if data at virtual address \( a \) is at physical address \( a' \) in \( P \)
    - \( MAP(a) = \emptyset \) if data at virtual address \( a \) not in physical memory
      - Either invalid or stored on disk

VM Address Translation: Hit

VM Address Translation: Miss
VM Address Translation

Parameters
- \( P = 2^p \): page size (bytes).
- \( N = 2^n \): Virtual address limit
- \( M = 2^m \): Physical address limit

Page offset bits don’t change as a result of translation.

Address Translation via Page Table

Page Tables

Page Table Operation

Computing Physical Address

Page Table Operation

Checking Protection
**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

**Speeding up 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

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

**Simple Memory System Example**

**Address Translation with a TLB**

**Simple Memory System Page Table**
- Only show first 16 entries

**Simple Memory System TLB**
- 16 entries
- 4-way associative
Simple Memory System Cache

Cache

- 16 lines
- 4-byte line size
- Direct mapped

Address Translation Example #1

Virtual Address 0x03D4

Physical Address

Address Translation Example #2

Virtual Address 0x0B8F

Physical Address

Address Translation Example #3

Virtual Address 0x0040

Physical Address

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!
- 2^30 or 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

Main Themes

Programmer's View

- Large "flat" address space
- Can allocate large blocks of contiguous addresses
- Process "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
- Continuously switching among processes
- Especially when one must wait for resource
  - E.g., disk I/O to handle page fault