CS 201
Computer Systems Programming II – Memory Management

Notes based on robboy’s notes and instructors notes from B&O’s web site:
http://csapp.cs.cmu.edu

Review… VM address translation

- Remember: all processes have their own virtual address space
- In addition to: memory protection, relocation, swapping, ...

Processor

On-chip MMU

Main Memory

virtual address

physical address

Review… ia32

Suppose a computer has 16-bit virtual addresses, 16-bit physical addresses, a page size of 64 bytes, and two-level page tables, like a pentium.
- How many bits is the VPO?
- How many entries in a page table?
- How many bits is VPN0?
- How many bits is VPN1?
Questions

- If for every memory reference, we had to do a two-level table lookup, then every memory reference would actually involve three memory references
  - Page directory, page table, and memory containing data

- Would this be good for performance?

- How can it be optimized?
Translation Lookaside Buffer

What is the TLB

- The TLB is an on-chip cache of page mappings
- The TLB converts a virtual page address to a physical page address
- On a TLB hit, the TLB delivers a physical page address
  - Bypasses the page tables entirely
- Even on a TLB miss, the page table entries may be in the memory cache.

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.

CS 201
Computer Systems Programming II – Virtual Memory

Notes based on robbys notes and instructors notes from B&O’s web site: http://csapp.cs.cmu.edu
Motivation #1: DRAM a “Cache” for Disk

- Full address space is quite large:
  - 32-bit addresses: \( \sim 4,000,000,000 \) (4 billion) bytes
  - 64-bit addresses: \( \sim 16,000,000,000,000 \) (16 quintillion) bytes
- Disk storage is \( \sim 300X \) 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>DRAM</th>
<th>Disk</th>
<th>DRAM</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 GB</td>
<td>$200</td>
<td>$110</td>
<td>4 MB</td>
<td>$500</td>
</tr>
<tr>
<td>80 GB</td>
<td>$33,000</td>
<td>$110</td>
<td>80 GB</td>
<td>$110</td>
</tr>
</tbody>
</table>

Levels in Memory Hierarchy

- CPU
- Registers
- Memory
- Page Table
- Disk

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

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 suspends, others can resume
    - 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 as assigned to physical memory

Virtual Memory = Swapping

- Pieces of processes are swapped in and out.
- The granularity is the page, not the whole process
- Pages are not in memory until needed
  - “Demand Paging”
  - Pull pages in on demand; i.e., when accessed
Page Replacement algorithms

- Analogous to cache line replacement.
- A complex topic.
  - Beyond the scope of this class
  - A popular topic with computer scientists because it lends itself to research.

Motivation #3: Protection

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

<table>
<thead>
<tr>
<th>Process i:</th>
<th>Read?</th>
<th>Write?</th>
<th>Physical Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>VP 0:</td>
<td>Yes</td>
<td>No</td>
<td>PP 9</td>
</tr>
<tr>
<td>VP 1:</td>
<td>Yes</td>
<td>Yes</td>
<td>PP 4</td>
</tr>
<tr>
<td>VP 2:</td>
<td>No</td>
<td>No</td>
<td>XXXXXXX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process j:</th>
<th>Read?</th>
<th>Write?</th>
<th>Physical Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>VP 0:</td>
<td>Yes</td>
<td>Yes</td>
<td>PP 6</td>
</tr>
<tr>
<td>VP 1:</td>
<td>Yes</td>
<td>No</td>
<td>PP 9</td>
</tr>
<tr>
<td>VP 2:</td>
<td>No</td>
<td>No</td>
<td>XXXXXXX</td>
</tr>
</tbody>
</table>

VM Address Translation

- Virtual Address Space
  - $V = \{0, 1, \ldots, N-1\}$
- Physical Address Space
  - $P = \{0, 1, \ldots, M-1\}$
  - $M < N$
- Address Translation
  - $MAP: V \rightarrow P \cup \{\emptyset\}$
  - For virtual address $a$:
    - $MAP(a) = a'$ if data at virtual address $a$ 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

- Processor
- Hardware Addr Trans Mechanism
- Main Memory

Virtual address → physical address

part of the on-chip memory mgmt unit (MMU)

VM Address Translation: Miss

- Processor
- Hardware Addr Trans Mechanism
- Main Memory
- Secondary memory

Virtual address

physical address

part of the on-chip memory mgmt unit (MMU)

page fault

page fault handler

OS performs this transfer (only if miss)

VM Address Translation

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

- Virtual Page Number
  - m-1 p p-1 0
  - virtual page number
  - page offset

- Physical Memory
  - m-1 p p-1 0
  - physical page number
  - page offset

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

Page Tables

- Memory resident page table (physical page or disk address)
- Disk Storage (swap file or regular file system file)
Address Translation via Page Table

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

Page Table Operation
- 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

Page Table Operation
- 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