Virtual Memory
CSE 361S

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.

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 ~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

4 MB: ~$500
10 GB: ~$200
80 GB: ~$110

Levels in Memory Hierarchy

<table>
<thead>
<tr>
<th></th>
<th>Register</th>
<th>Cache</th>
<th>Memory</th>
<th>Disk Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>32 B</td>
<td>32 KB-4MB</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>$/MB</td>
<td>$125/MB</td>
<td>$0.20/MB</td>
<td>$0.20/MB</td>
<td>$0.001/MB</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

Object Name = X?

-7-

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

Object Name

-8-

A System with Physical Memory Only

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

CPU

Memory

Physical Addresses

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

-9-

A System with Virtual Memory

Examples:
- workstations, servers, modern PCs, etc.

CPU

Memory

Page Table

Virtual Addresses

Physical Addresses

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

-10-

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.

Before fault

After fault

Servicing a Page Fault

Processor Signals Controller
- Read block of length P starting at disk address X and store starting at memory address Y
- Direct Memory Access (DMA)
- Under control of I/O controller

1/0 Controller Signals Completion
- Interrupt processor
- OS resumes suspended process

(1) Initiate Block Read
(2) DMA Transfer
(3) Read Done

-11-

Page 2
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

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

Address Translation
- MAP: \( V \rightarrow P \cup \{\emptyset\} \)
- For virtual address \( a \):
  - MAP\((a) = a' \) if data at virtual address \( a \) 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**
- \( P = 2^n \) = page size (bytes).
- \( N = 2^n \) = Virtual address limit
- \( M = 2^m \) = Physical address limit

\[ \begin{array}{c|c|c}
\text{virtual page number} & \text{physical page number} & \text{valid bit} \\
\hline
n-1 & p-1 & 0 \\
\end{array} \]

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

---

**Page Tables**

**Address Translation via Page Table**

- VPN acts as table index
- If valid = 0 then page not in memory

\[ \begin{array}{c|c|c}
\text{valid access} & \text{physical page number (PPN)} & \text{page offset} \\
\hline
& m-1 & p-1 \\
\end{array} \]

---

**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

\[ \begin{array}{c|c|c}
\text{valid access} & \text{physical page number (PPN)} & \text{page offset} \\
\hline
& m-1 & p-1 \\
\end{array} \]

---

**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

\[ \begin{array}{c|c|c}
\text{valid access} & \text{physical page number (PPN)} & \text{page offset} \\
\hline
& m-1 & p-1 \\
\end{array} \]
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

Addressing
- 14-bit virtual addresses
- 12-bit physical address
- Page size = 64 bytes

Simple Memory System Page Table

- Only show first 16 entries

Simple Memory System TLB

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

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

<table>
<thead>
<tr>
<th>Index</th>
<th>Tag</th>
<th>Valid</th>
<th>D0</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
<th>Index</th>
<th>Tag</th>
<th>Valid</th>
<th>D0</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>19</td>
<td>1</td>
<td>06</td>
<td>11</td>
<td>23</td>
<td>11</td>
<td>0</td>
<td>24</td>
<td>1</td>
<td>34</td>
<td>1</td>
<td>8</td>
<td>46</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
<td>9</td>
<td>00</td>
<td>0</td>
<td>00</td>
<td>00</td>
<td>1</td>
<td>15</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>2</td>
<td>18</td>
<td>1</td>
<td>00</td>
<td>06</td>
<td>06</td>
<td>06</td>
<td>2</td>
<td>00</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>3</td>
<td>36</td>
<td>0</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>3</td>
<td>00</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>1</td>
<td>00</td>
<td>06</td>
<td>06</td>
<td>06</td>
<td>4</td>
<td>12</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>5</td>
<td>00</td>
<td>1</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>5</td>
<td>15</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>6</td>
<td>34</td>
<td>0</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>6</td>
<td>18</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>1</td>
<td>00</td>
<td>06</td>
<td>06</td>
<td>06</td>
<td>7</td>
<td>30</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>

Address Translation Example #1

Virtual Address: 0x03D4

- TLBI
- TLBT
- TLB Hit?
- Page Fault?
- PPN:

Physical Address

- CT
- CI
- CO
- PPN

Address Translation Example #2

Virtual Address: 0x0B8F

- TLBI
- TLBT
- TLB Hit?
- Page Fault?
- PPN:

Physical Address

- CT
- CI
- CO
- PPN

Address Translation Example #3

Virtual Address: 0x0040

- TLBI
- TLBT
- TLB Hit?
- Page Fault?
- PPN:

Physical Address

- CT
- CI
- CO
- PPN

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 * 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
- 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
  - Continuously switching among processes
  - Especially when one must wait for resource
    - E.g., disk I/O to handle page fault