A System Using Physical Addressing

Used by many digital signal processors and embedded microcontrollers in devices like phones and PDAs.

A System Using Virtual Addressing

A linear address space is an ordered set of contiguous nonnegative integer addresses:

\{0, 1, 2, 3, \ldots \}

A virtual address space is a set of \( N = 2^n \) virtual addresses:

\{0, 1, 2, \ldots, N-1\}

A physical address space is a set of \( M = 2^m \) (for convenience) physical addresses:

\{0, 1, 2, \ldots, M-1\}

In a system based on virtual addressing, each byte of main memory has a virtual address and a physical address.
Why Virtual Memory?

(1) VM uses main memory efficiently
- Main memory is a cache for the contents of a virtual address space stored on disk.
- Keep only active areas of virtual address space in memory.
- Transfer data back and forth as needed.

(2) VM simplifies memory management
- Each process gets the same linear address space.

(3) VM protects address spaces
- 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.

(1) VM as a Tool for Caching

Virtual memory is an array of N contiguous bytes stored on disk.
The contents of the array on disk are cached in physical memory (DRAM cache)

DRAM Cache Organization

DRAM cache organization driven by the enormous miss penalty
- DRAM is about 10x slower than SRAM
- Disk is about 100,000x slower than a DRAM

DRAM cache properties
- Large page (block) size (typically 4-8 KB)
- Fully associative
  - Any virtual page can be placed in any physical page
- Highly sophisticated replacement algorithms
- Write-back rather than write-through

Page Tables

A page table is an array of page table entries (PTEs) that maps virtual pages to physical pages.
- Kernel data structure in DRAM
- Physical page number or disk address
- Memory resident page table (DRAM)
**Page Hits**

A *page hit* is a reference to a VM word that is in physical (main) memory.

**Page Faults**

A *page fault* is caused by a reference to a VM word that is not in physical (main) memory.

- Example: A instruction references a word contained in VP 3, a miss that triggers a page fault exception

---

**Servicing a Page Fault**

1. Processor signals controller
   - Read block of length P starting at disk address X and store starting at memory address Y
   - When the offending instruction restarts, it executes normally, without generating an exception

2. Read occurs
   - Direct Memory Access (DMA)
   - Under control of I/O controller

3. Controller signals completion
   - Interrupt processor
   - OS resumes suspended process
Allocating Virtual Pages

Example: Allocating new virtual page VP5
- Kernel allocates VP 5 on disk and points PTE 5 to this new location.

Virtual memory works because of locality.
At any point in time, programs tend to access a set of active virtual pages called the **working set**.
- Programs with better temporal locality will have smaller working sets.

If working set size < main memory size
- Good performance after initial compulsory misses.

If working set size > main memory size
- **Thrashing**: Performance meltdown where pages are swapped (copied) in and out continuously.

Locality to the Rescue

Simplifying Sharing and Allocation

Sharing code and data among processes
- Map virtual pages to the same physical page (PP 7)

Memory allocation
- Virtual page can be mapped to any physical page
Simplifying Linking and Loading

Linking
- Each program has similar virtual address space
- Code, stack, and shared libraries always start at the same address.

Loading
- `execve()` maps PTEs to the appropriate location in the executable binary file.
- The `.text` and `.data` sections are copied, page by page, on demand by the virtual memory system.

Memory invisible to user code

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

Physical Address Space
- \( P = \{0, 1, \ldots, M-1\} \)
- \( M < N \) (usually, but \( \geq 4 \) Gbyte on an IA32 possible)

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 as a Tool for Memory Protection

Extend PTEs with permission bits.
Page fault handler checks these before remapping.
- If violated, send process SIGSEGV (segmentation fault)

Address Translation with a Page Table

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

Physical Address Space
- \( P = \{0, 1, \ldots, M-1\} \)
- \( M < N \) (usually, but \( \geq 4 \) Gbyte on an IA32 possible)

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
Address Translation: Page Hit

1) Processor sends virtual address to MMU
2) MMU fetches PTE from page table in memory
3) MMU sends physical address to L1 cache
4) L1 cache sends data word to processor

Address Translation: Page Fault

1) Processor sends virtual address to MMU
2) MMU fetches PTE from page table in memory
3) MMU sends PTE to page fault exception handler
4) Valid bit is zero, so MMU triggers page fault exception
5) Handler identifies victim, and if dirty pages it out to disk
6) Handler pages in new page and updates PTE in memory
7) Handler returns to original process, restarting faulting instruction.

Integrating VM and Cache

Page table entries (PTEs) are cached in L1 like any other memory word.
- PTEs can be evicted by other data references
- PTE hit still requires a 1-cycle delay

Solution: Cache PTEs in a small fast memory in the MMU.

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

Speeding up Translation with a TLB
A TLB hit eliminates a memory access.

A TLB miss incurs an additional memory access (the PTE).
Fortunately, TLB misses are rare. Why?

Simple Memory System Example

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

<table>
<thead>
<tr>
<th>VPN</th>
<th>PPN</th>
<th>Valid</th>
<th>VPN</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>28</td>
<td>1</td>
<td>08</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>–</td>
<td>0</td>
<td>09</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>02</td>
<td>33</td>
<td>1</td>
<td>0A</td>
<td>09</td>
<td>1</td>
</tr>
<tr>
<td>03</td>
<td>02</td>
<td>1</td>
<td>0B</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>04</td>
<td>–</td>
<td>0</td>
<td>0C</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>05</td>
<td>16</td>
<td>1</td>
<td>0D</td>
<td>2D</td>
<td>1</td>
</tr>
<tr>
<td>06</td>
<td>–</td>
<td>0</td>
<td>0E</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>07</td>
<td>–</td>
<td>0</td>
<td>0F</td>
<td>0D</td>
<td>1</td>
</tr>
</tbody>
</table>

Simple Memory System Page Table

- Only show first 16 entries (out of 256)
Simple Memory System TLB

- 16 entries
- 4-way associative

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>03</td>
<td>-</td>
<td>0</td>
<td>09</td>
<td>D</td>
<td>1</td>
<td>00</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>03</td>
<td>2D</td>
<td>1</td>
<td>02</td>
<td>-</td>
<td>0</td>
<td>04</td>
<td>-</td>
<td>0A</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>-</td>
<td>0</td>
<td>08</td>
<td>-</td>
<td>0</td>
<td>06</td>
<td>-</td>
<td>03</td>
</tr>
<tr>
<td>3</td>
<td>07</td>
<td>-</td>
<td>0</td>
<td>03</td>
<td>D</td>
<td>1</td>
<td>0A</td>
<td>34</td>
<td>1</td>
</tr>
</tbody>
</table>

TLB

VPO

VPN

Address Translation Example #1

Virtual Address 0x03D4

VPN 0x0F

TLBI 3

TLBT 0x03

TLB Hit? Y

Page Fault? NO

PPN 0x0D

Physical Address

0 0 0 0 0 1 1 1 1 0 1 0 1 0 0

Address Translation Example #2

Virtual Address 0x0B8F

VPN 0x2E

TLBI 2

TLBT 0x0B

TLB Hit? NO

Page Fault? YES

PPN: TBD

Physical Address

0 0 1 0 1 1 1 0 1 0 0 1 1 1 1 1
Address Translation Example #3

Virtual Address 0x0020

<table>
<thead>
<tr>
<th>13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0 0 0 1 0 0 0 0 0</td>
</tr>
</tbody>
</table>

VPN ___ TLBI ___ TLBT ____ TLB Hit? __ Page Fault? __ PPN: __

Physical Address

<table>
<thead>
<tr>
<th>11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 0 0 0 1 0 0 0 0 0</td>
</tr>
</tbody>
</table>

Offset ___ CI___ CT ____ Hit? __ Byte: __

VPO

VPN 0x00 TLBI 0 TLBT 0x00 TLB Hit? NO Page Fault? NO PPN 0x28

Multi-Level Page Tables

Given:
- 4KB (2^12) page size
- 48-bit address space
- 4-byte PTE

Problem:
- Would need a 256 GB page table!
  - 2^48 * 2^-12 * 2^2 = 2^38 bytes

Common solution:
- Multi-level page tables
- Example: 2-level page table
  - Level 1 table: each PTE points to a page table (memory resident)
  - Level 2 table: Each PTE points to a page (paged in and out like other data)

A Two-Level Page Table Hierarchy

Level 1
page table

Level 2
page tables

Virtual memory

VP 0

2K allocated VM pages for code and data

VP 1023

6K unallocated VM pages

VP 2047

1023 unallocated pages

Gap

512K unallocated pages

1023 null PTEs

PTE 0

PTE 1

PTE 2 (null)

PTE 3 (null)

PTE 4 (null)

PTE 5 (null)

PTE 6 (null)

PTE 7 (null)

PTE 8

(1K - 9)

null PTEs

PTE 1023

1023 null PTEs

PTE 1023

PTE 0

PTE 1023

PTE 1023

VP 9215

1 allocated VM page for the stack

Translating with a k-level Page Table


Summary

Programmer's View of Virtual Memory
- Each process has its own private linear address space
- Cannot be corrupted by other processes

System View of Virtual Memory
- Uses memory efficiently by caching virtual memory pages stored on disk.
  - Efficient only because of locality
- Simplifies memory management in general, linking, loading, sharing, and memory allocation in particular.
- Simplifies protection by providing a convenient interpositioning point to check permissions.