Memory Management Part 1
The Address-Space Concept

- Protect processes from one another
- Protect the OS from user processes
- Provide efficient management of available storage
Memory Fence

User Area

OS
Base and Bounds Registers

Base register

Bounds

Base register

Bounds

Bounds
Swapping

User Area

OS
Overlays

- Overlay
- Resident
Structuring Virtual Memory

• Paging
  – divide the address space into fixed-size pages

• Segmentation
  – divide the address space into variable-size segments (typically each corresponding to some logical unit of the program, such as a module or subroutine)
Paging

• Map fixed-size pages into memory (into page frames)
• Many hardware mapping techniques
  – page tables
  – translation lookaside buffers
# Page Tables

<table>
<thead>
<tr>
<th>Page #</th>
<th>Offset</th>
</tr>
</thead>
</table>

![Diagram of Page Tables](image)

- **Virtual Address**
- **Page Frame #**
- **V**
- **M**
- **R**
- **Prot**
Page-Table Size

• Consider a full $2^{32}$-byte address space
  – assume 4096-byte ($2^{12}$-byte) pages
  – 4 bytes per page table entry
  – the page table would consist of $2^{32}/2^{12}$ (= $2^{20}$) entries
  – its size would be $2^{22}$ bytes (or 4 megabytes)
Forward-Mapped Page Table

L1 Page # | L2 Page # | Offset

L1 Page table | L2 Page tables | Page frame
IA32 Paging

CR3

Page directory table

Page table

Page
Quiz 1

Suppose a process on an IA32 has exactly one page residing in real memory. What is the total number of combined pages of page-directory table and page tables required to map this page?

a) 1  
b) 2  
c) 4  
d) 8
x86-64 Address Space

- OS kernel: 0xffffffffffffffff to 0xffffffff800000000000000
- Illegal: 0xffffffff0000000000000000 to 0xffffffff7fffffffffff
- User: 0x00007fffffffffff to 0x0000000000000000

2^47 bytes for each section.
x86-64 Virtual Address Format 3

Unused

Page map table

Page directory pointer table

1GB page
Why Multiple Page Sizes?

• External fragmentation
  – for region composed of 4KB pages, average external fragmentation is 2KB
  – for region composed of 1GB pages, average external fragmentation is 512MB

• Page-table overhead
  – larger page sizes have fewer page tables
    - less overhead in representing mappings
      • both in memory and in cache
Linear Page Table

2 21 9
S VPN Offset

Space x Page Table

Page
VAX Linear Page Translation

VA: 00 VPN Offset
00 BR

PTEA: 10 VPN Offset
10 BR

10 PT

00 PT

10 AS

00 AS
• VAX architecture introduced in 1978
  – memory cost $40,000/MB
    - 3.8¢/byte
      (.475¢/bit)
Linear Page-Table Management

• 00 and 01 page tables each require contiguous locations in 10 space
  – with 512-byte pages, 8MB each:
    - maximum of 128 such page tables
    - (need room for other things, e.g. OS)
• Reduce size requirements with partial page tables
  – length registers constrain size of each space
Traditional Unix with Linear PTs

01 AS

00 AS

stack

dynamic

bss

data

text
Quiz 2

Suppose the page size is 512 bytes ($2^9$) and each page-table entry requires 4 bytes. How many pages of page-table entries are required to map 1 megabyte ($2^{20}$) of address space?

a) 4  
b) 8  
c) 16  
d) 32
• Limit size of 00 space to 1 MB
  – requires 16-page 00 page table in 10 virtual memory
    - requires 16 entries in 10 page table
• Same requirements if 01 space limited to 1 MB
• What are real-memory requirements?
  – 10 page table resides in real memory
  – at least one page of real memory must be allocated for each of 00 and 01 page tables
  – minimum real memory is 1152 bytes
    - $43.95 in 1978
• Requires sufficient 10 page-table entries to map almost all of 00 and 01 space
  – $2^{14}$ 10 page-table entries for each space
    - requiring 64KB each, 128KB total
    - $5000 in 1978

  - <1¢/process today
    • who cares?
    • increase address space from $2^{32}$ to $2^{64}$
      – 4,294,967,296-fold increase
      – significant …
Hashed Page Tables

Hashed Page Tables use a hash function to map page frames to page table entries. The hash function takes the page number and offset as input and produces a hash value. This hash value is then used to index into the page table, allowing for fast lookup of the corresponding page frame. The page table entry contains the necessary information to map virtual addresses to physical page frames. This approach is efficient for managing memory addresses in operating systems, providing a balanced trade-off between memory usage and performance.
Clustered Page Tables

<table>
<thead>
<tr>
<th>Page #</th>
<th>Offset</th>
</tr>
</thead>
</table>

Virtual Address

Hash Table

Hash

Tag
Link
Entry
Entry
Entry
Entry
Entry
Entry
...

Pages
Inverted Page Tables

<table>
<thead>
<tr>
<th>Page #</th>
<th>Offset</th>
<th>Virtual Address</th>
</tr>
</thead>
</table>

Hash Table

Page Frame Table
Translation Lookaside Buffers

<table>
<thead>
<tr>
<th>14</th>
<th>6</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>Key</td>
<td>Offset</td>
</tr>
</tbody>
</table>

- Tag
- Page Frame #
- Tag
- Page Frame #
- Tag
- Page Frame #
- ...
- ...
- Tag
- Page Frame #
TLBs and Multiprocessors
// shooter code
for all processors i sharing address space
  interrupt(i);
for all processors i sharing address space
  while (noted[i] == 0)
    ;
modify_page_table();
update_or_flush_tlb();
done[me] = 1;

// shootee i interrupt handler
receive_interrupt_from_processor j
  noted[i] = 1
while (done[j] == 0)
  ;
flush_tlb()
IA-64 Address Translation

- TLB
  - software-managed

- Virtual Hash Page Table (VHPT)
  - per-region linear page table
  - single large hashed page table
Translation: TLB

- region reg
- page number
- offset
- TLB
- offset
Translation: TLB Miss

region reg

Hash Function

Virtual Hash Page Table
### Virtual Machines Meet Virtual Memory

<table>
<thead>
<tr>
<th>Virtual machine’s page table</th>
<th>Virtual virtual memory</th>
<th>VMM’s page table</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0 i</td>
</tr>
<tr>
<td>1</td>
<td>i</td>
<td>1 3</td>
</tr>
<tr>
<td>2</td>
<td>i</td>
<td>2 1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Virtual real memory</th>
<th>Real memory</th>
<th>Shadow page table</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 i</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 i</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 1</td>
</tr>
</tbody>
</table>
Paravirtualization to the Rescue

virtual memory

virtual memory

real memory

Direct translation
Hardware to the Rescue

virtual memory

0 1
1 i
2 i
3 2

virtual real memory

Extended Page Tables

0 i
1 3
2 1
3 2

real memory
x86 Paging with EPT

10 bits  |  10 bits  |  12 bits

CR3 → Page Directory (pd) → Page Table (pt) → EPTP → Page
Quiz 3

We’d like to virtualize EPT. Assume that setting EPTP causes a VMexit if done on a VMM that’s not running in real ring -1. What does the VMM running at level 0 do when it receives such a VMexit from a VMM running at level 1?

a) nothing: the EPT mechanism is virtualized by the hardware

b) it sets EPTP to point to the composition of the page tables mapping the level 1 VM (on which the level 1 VMM runs) and the page tables pointed to by the value being attempted to be put in EPT

c) something else