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
Swapping
Overlays

Overlay

Resident
Virtual Memory

Process 1

Process 2

Process 3

Memory

Disk
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

Virtual Address

Page # | Offset

V | M | R | Prot | Page Frame #
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

<table>
<thead>
<tr>
<th>L1 Page #</th>
<th>L2 Page #</th>
<th>Offset</th>
</tr>
</thead>
</table>

- **L1 Page table**
- **L2 Page tables**
- **Page frame**
IA32 Paging

CR3

Page directory table

Page table

Page
x86-64 Virtual Address Format 2

- Page map table
- Page directory pointer table
- Page directory table

2MB page
x86-64 Virtual Address Format 3

63 47 38 29 0

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
Address Space

- **OS kernel**: 0xffffffffffffffff
- **Illegal**: 0xffffffff80000000000000
  - 2⁴⁷ bytes
- **User**: 0x00000800000000000
  - 2⁶⁴ – 2⁴⁸ bytes
- **User**: 0x000007ffffffffffff
  - 2⁴⁷ bytes

Address Space Layout:

- 0xffffffffffffffff
- 0xffffffff80000000000000
- 0xffffffff7fffffffffffff
- 0x00000800000000000
- 0x000007ffffffffffff
- 0x0000000000000000
- 0x0000000000000000

2⁴⁷ bytes
2⁶⁴ – 2⁴⁸ bytes
2⁴⁷ bytes
Linear Page Table

Space

Page

Space x Page Table
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
      (4.75¢/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
• 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
Modern Unix

01 AS

stack1
stack2
stack3
mapped file
mapped file
dynamic
bss
data
text

00 AS
$\%

• 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 virtual addresses to page frames. The hash function takes the virtual address as input and produces an index into the page table. This index is used to look up the corresponding page frame in the page table.

The page table is divided into pages, each containing a set of page table entries (PTEs). Each PTE contains the following fields:

- Tag: A unique identifier for the page frame.
- Link: A link to the next PTE in the chain.
- Entry: The actual page frame contents.

The hash function is applied to the virtual address to determine which page table page to access. Once the correct page table page is found, the hash function is applied again to find the correct PTE within that page.

This mechanism allows for efficient access to page frames, as the hash function can be computed quickly, and the page table is organized to minimize the number of PTEs that need to be searched.

The diagram shows the flow of data from the virtual address to the page frame, illustrating how the hash function is used to compute the correct page table and PTE indices.
Clustered Page Tables

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

Hash Table

Pages

Hash

Tag
Link
Entry
Entry
Entry
Entry
Entry
Entry
Entry
Entry
Inverted Page Tables

- Page #
- Offset

Virtual Address

Hash

Hash Table

Page Frame Table
Translation Lookaside Buffers

<table>
<thead>
<tr>
<th>Tag</th>
<th>Key</th>
<th>Offset</th>
</tr>
</thead>
</table>

- Tag
- Page Frame #

Tag
Page Frame #

Tag
Page Frame #

Tag
Page Frame #

Tag
Page Frame #

Tag
Page Frame #
TLBs and Multiprocessors

Processor 1

TLB

Processor 2

TLB

Process
TLB Shootdown Algorithm

// 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

TLB

page number

offset

offset
Translation: TLB Miss

<table>
<thead>
<tr>
<th>region reg</th>
<th>Hash Function</th>
</tr>
</thead>
</table>

Virtual Hash Page Table

page number | offset
Virtual Machines Meet Virtual Memory

Virtual machine’s page table

Virtual virtual memory

VMM’s page table

Virtual real memory

Real memory

Shadow page table
Paravirtualization to the Rescue

virtual memory

real memory

Direct translation
Hardware to the Rescue

virtual memory

0
1 i
2 i
3 2

virtual memory

Extended Page Tables

real memory
x86 Paging with EPT

- CR3
- Page Directory (pd)
- Page Table (pt)
- EPTP
- Page