Virtual Memory
Process Abstraction, Part 2: Private Address Space

Motivation: why not direct physical memory access?
Address translation with pages
Extra benefits: sharing and protection

Memory as a contiguous array of bytes is a lie! Why?

Problem 1: memory management

Process 1
Process 2
Process 3
... Process n

Stack
Heap
Code
Globals...

What goes where?

Also:
Context switches must swap out entire memory contents.
Isn’t that expensive?

Problem 2: capacity

64-bit addresses can address several exabytes
(18,446,744,073,709,551,616 bytes)

Physical main memory offers a few gigabytes
(e.g. 8,589,934,592 bytes)

(To scale with 64-bit address space, you can’t see it.)
Problem 3: protection

Problem 4: sharing

Solution: Virtual Memory (address indirection)

Indirection (it’s everywhere!)

Tangent: indirection everywhere

"Any problem in computer science can be solved by adding another level of indirection."
—David Wheeler, inventor of the subroutine, or Butler Lampson

Another Wheeler quote? “Compatibility means deliberately repeating other people’s mistakes.”
Virtual addressing and address translation

**Memory Management Unit**
-
- translates virtual address to physical address

**CPU Chip**

- Virtual address (VA) to Physical address (PA)

**Main memory**

- 0
- 1
- ... (M-1)

**Data**

Page-based mapping

**fixed-size, aligned pages**
- page size = power of two

**Physical Address Space**

- 0
- 1
- ... (2^n - 1)

**Virtual Address Space**

- 0
- 1
- ... (2^n - 1)

Map virtual pages onto physical pages.

Some virtual pages do not fit!
Where are they stored?

Cannot fit all virtual pages! Where are the rest stored?

Virtual memory: cache for disk?

**SRAM**
- L1 cache
- Throughput: 16 B/cycle
- Latency: 3 cycles

**DRAM**
- L1 unified cache
- Throughput: 8 B/cycle
- Latency: 14 cycles

**Main Memory**
- L1 D-cache
- Throughput: 2 B/cycle
- Latency: 100 cycles
- Memory miss penalty (latency): 10,000x

**Disk**
- ~500 GB
- solid-state “flash” or spinning magnetic platter.

Example system

- CPU
- Reg
- Throughput: 2 B/30 cycles
- Memory miss penalty (latency): 33x

<table>
<thead>
<tr>
<th>Cache miss penalty (latency): 33x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput: 16 B/cycle</td>
</tr>
<tr>
<td>Latency: 3 cycles</td>
</tr>
<tr>
<td>Throughput: 8 B/cycle</td>
</tr>
<tr>
<td>Latency: 14 cycles</td>
</tr>
<tr>
<td>Throughput: 2 B/cycle</td>
</tr>
<tr>
<td>Latency: 100 cycles</td>
</tr>
<tr>
<td>Memory miss penalty (latency): 10,000x</td>
</tr>
<tr>
<td>Throughput: 1 B/30 cycles</td>
</tr>
</tbody>
</table>

Not drawn to scale!
Virtual memory benefits:

**Simple address space allocation**

Process needs private *contiguous* address space.

**Protection:**

All accesses go through translation. Impossible to access physical memory not mapped in virtual address space.

**Sharing:**

Map virtual pages in separate address spaces to same physical page (e.g., execute-only library code: libc).
Virtual memory benefits:
Memory permissions

MMU checks on every access.
Exception if not allowed.

Summary: virtual memory

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 (due to locality) by caching virtual memory pages
Simplifies memory management and sharing
Simplifies protection -- easy to interpose and check permissions
More goodies:
• Memory-mapped files
• Cheap fork() with copy-on-write pages (COW)