Virtual Memory
Process Abstraction, Part 2: Private Address Space

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

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

Solution: Virtual Memory (address indirection)

Private virtual address space per process.
Single physical address space managed by OS/hardware.

Virtual Addressing and Address Translation

Physical addresses are invisible to programs.
Page-based Mapping

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

Map virtual pages onto physical pages.

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

Page Table
array of *page table entries* (PTEs)
mapping virtual page to where it is stored

Physical pages
(Physical memory)

PTE 0
Valid
0 null
1
1
0
1
0 null
0
PTE 7

Swap space
(Disk)

How many page tables are in the system?

Design for a Slow Disk: Exploit Locality

Virtual Memory Address Space

on disk

Physical Memory Address Space

Address Translation with a Page Table

Virtual address (VA)

Base address of current process's
page table

Swap page mapped
to physical page?

Yes = Page Hit
**Page Hit:** virtual page in memory

**Page Fault:** exceptional control flow

Process accessed virtual address in a page that is not in physical memory.

Process

User Code

OS exception handler

`movl` exception: page fault

Load page into memory

return

Returns to faulting instruction: `movl` is executed again!

**Page Fault: 1. page not in memory**

What now?

OS handles fault

**Page Fault: 2. OS evicts another page.**

"Page out"
**Page Fault:** OS loads needed page.

```
Virtual Page Number

Physical Page Number
or disk address

PTE 0
Valid
0 null
1 On disk
1 PP 1
1 PP 0
1 PP 3
0 null
0 On disk
1 PP 2

PTE 7

Valid
0 null
1 On disk
```

Finally:
Re-execute faulting instruction.
Page hit!

### Terminology

- **context switch**
- **page in**
- **page out**
- **thrash**

Useful for "real life" too.

### Address Translation: Page Hit

```
CPU Chip

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>VA</td>
<td>PTE</td>
</tr>
<tr>
<td>MMU</td>
<td>PTEA</td>
<td>PA</td>
</tr>
<tr>
<td>Cache/Memory</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

### Address Translation: Page Fault

```
CPU Chip

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>VA</td>
<td>PTE</td>
<td>PTEA</td>
<td>RA</td>
</tr>
<tr>
<td>MMU</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cache/Memory</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
Exception

<table>
<thead>
<tr>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page fault handler</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
Cache/Memory

<table>
<thead>
<tr>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Victim page</td>
<td></td>
<td></td>
</tr>
<tr>
<td>New page</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Disk
How fast is translation?

How many physical memory accesses are required to complete one virtual memory access?

Translation Lookaside Buffer (TLB)

Small hardware cache in MMU just for page table entries e.g., 128 or 256 entries

Much faster than a page table lookup in memory.

In the running for "un/classiest name of a thing in CS"

A TLB hit eliminates a memory access

A TLB miss incurs an additional memory access (the PTE)
Fortunately, TLB misses are rare. Does a TLB miss require disk access?

Simple Memory System Example (small)

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

Simulate accessing these virtual addresses on the system: 0x03D4, 0x0B8F, 0x0020
Simple Memory System Page Table

Only showing first 16 entries (out of 256 = 2^8)

<table>
<thead>
<tr>
<th>virtual page #</th>
<th>TLB index</th>
<th>TLB tag</th>
<th>TLB Hit?</th>
<th>physical page #</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 28 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01 – 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>02 33 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>03 02 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>04 – 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>05 16 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>06 – 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07 – 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What about a real address space? Read more in the book…

Simple Memory System TLB

16 entries
4-way associative

TLB ignores page offset. Why?

Simple Memory System Cache

16 lines
4-byte block size
Physically addressed
Direct mapped

physical page number physical page offset