Virtual Memory

Reading

- Sections 5.4, 5.5, 5.6, 5.8, 5.10
The Memory Hierarchy

Managed by the compiler

Managed by the hardware

Managed by the operating system

Faster

Cheaper

Virtual to Physical Mapping

- Exploit program locality at page granularity
- Program can be larger than memory
- At any point in time, the program is in memory+disk
Virtual Memory

- Use main memory as a “cache” for secondary (disk) storage
  - Managed jointly by CPU hardware and the operating system (OS)

- Programs share main memory
  - Each gets a private virtual address space holding its frequently used code and data
  - Protected from other programs

- CPU and OS translate virtual addresses to physical addresses
  - VM “block” is called a page
  - VM translation “miss” is called a page fault

Address Translation

- Fixed-size pages (e.g., 4K)

```
Virtual address

<table>
<thead>
<tr>
<th>Virtual addresses</th>
<th>Physical addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>31 30 29 28 27</td>
</tr>
<tr>
<td></td>
<td>15 14 13 12 11 10 9 8</td>
</tr>
<tr>
<td></td>
<td>3 2 1 0</td>
</tr>
</tbody>
</table>
```

```
Physical address

<table>
<thead>
<tr>
<th>Physical addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>29 28 27</td>
</tr>
<tr>
<td>15 14 13 12 11 10 9 8</td>
</tr>
<tr>
<td>3 2 1 0</td>
</tr>
</tbody>
</table>

Disk addresses
```

- Examples of translation
Address Translation: Concepts

- Offsets within the virtual page and corresponding physical page are the same
- We only need to translate the virtual page number (VPN) to the corresponding physical page number (PPN) also called *page frame* → effectively a *base address*

Page Tables

- Stores placement information
  - Array of page table entries, indexed by virtual page number
  - Page table register in CPU points to page table in physical memory
- If page is present in memory
  - Page table entry (PTE) stores the physical page number
  - Plus other status bits (referenced, dirty, ...)
- If page is not present
  - PTE can refer to location in swap space on disk
Translation Using a Page Table

Page table register

<table>
<thead>
<tr>
<th>Virtual address</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 30 29 28 27</td>
</tr>
<tr>
<td>15 14 13 12 11</td>
</tr>
<tr>
<td>10 9 8 ………… 3 2 1 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Virtual page number</th>
<th>Page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Valid</th>
<th>Physical page number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Page table

If 0 then page is not present in memory

<table>
<thead>
<tr>
<th>Physical page number</th>
<th>Page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Physical address

(9)

Page Fault Penalty

- On page fault, the page must be fetched from disk
  - Takes millions of clock cycles
  - Handled by OS code

- Try to minimize page fault rate
  - Fully associative placement
  - Smart replacement algorithms

(10)
Mapping Pages to Storage

To reduce page fault rate, prefer least-recently used (LRU) replacement:
- Reference bit (aka use bit) in PTE set to 1 on access to page
- Periodically cleared to 0 by OS
- A page with reference bit = 0 has not been used recently

Disk writes take millions of cycles:
- Write through is impractical
- Use write-back
- Dirty bit in PTE set when page is written
Caching PTEs: The Translation Lookaside Buffer (TLB)

- Keep a cache of most recently used PTEs
  - Each PTE corresponds to a “relatively” large part of memory
    - For example, a 16Kbyte page may have 4K instructions
  - A small set of PTEs can cover a large code segment
    - For example, 8 PTEs and 16 Kbyte pages corresponds to a program size of 32K instructions
- The TLB access time is comparable or better than cache access time
- Typically operates as a fully associative cache, but can be implemented as a set associative cache

Fast Translation Using a TLB

![Diagram of TLB and Page Table](image-url)
TLB Operation

- TLB size typically a function of the target domain
  - High end machines will have fully associative large TLBs
- PTE entries are replaced on a demand driven basis
- The TLB is in the critical path

TLB Misses

- If page is in memory
  - Load the PTE from memory and retry
  - Could be handled in hardware
    - Can get complex for more complicated page table structures
  - Or in software
    - Raise a special exception, with optimized handler
- If page is not in memory (page fault)
  - OS handles fetching the page and updating the page table
  - Then restart the faulting instruction
TLB Miss Handler

- TLB miss indicates one of
  - Page present, but PTE not in TLB
  - Page not present
- Must recognize TLB miss before destination register overwritten
  - Raise exception
- Handler copies PTE from memory to TLB
  - Then restarts instruction
  - If page not present, page fault will occur

Page Fault Handler

- Use faulting virtual address to find PTE
- Locate page on disk
- Choose page to replace
  - If dirty, write to disk first
- What about copies in the cache?
- Read page into memory and update page table
- Interaction with the operating system: make process runnable again
  - Restart from faulting instruction
### TLB and Cache Interaction

- **If cache tag uses physical address**
  - Need to translate before cache lookup

- **Alternative: use virtual address tag**
  - Complications due to aliasing
    - Different virtual addresses for shared physical address

- **Example problems**

---

### 2-Level TLB Organization

<table>
<thead>
<tr>
<th></th>
<th>Intel Nehalem</th>
<th>AMD Opteron X4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Virtual addr</strong></td>
<td>48 bits</td>
<td>48 bits</td>
</tr>
<tr>
<td><strong>Physical addr</strong></td>
<td>44 bits</td>
<td>48 bits</td>
</tr>
<tr>
<td><strong>Page size</strong></td>
<td>4KB, 2/4MB</td>
<td>4KB, 2/4MB</td>
</tr>
<tr>
<td><strong>L1 TLB (per core)</strong></td>
<td>L1 I-TLB: 128 entries for small pages, 7 per thread (2×) for large pages</td>
<td>L1 I-TLB: 48 entries L1 D-TLB: 48 entries Both fully associative, LRU replacement</td>
</tr>
<tr>
<td></td>
<td>L1 D-TLB: 64 entries for small pages, 32 for large pages</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Both 4-way, LRU replacement</td>
<td></td>
</tr>
<tr>
<td><strong>L2 TLB (per core)</strong></td>
<td>Single L2 TLB: 512 entries 4-way, LRU replacement</td>
<td>L2 I-TLB: 512 entries L2 D-TLB: 512 entries Both 4-way, round-robin LRU</td>
</tr>
<tr>
<td><strong>TLB misses</strong></td>
<td>Handled in hardware</td>
<td>Handled in hardware</td>
</tr>
</tbody>
</table>
Memory Protection

- Different tasks can share parts of their virtual address spaces
  - But need to protect against errant access
  - Requires OS assistance

- Hardware support for OS protection
  - Privileged supervisor mode (aka kernel mode)
  - Privileged instructions
  - Page tables and other state information only accessible in supervisor mode
  - System call exception (e.g., syscall in MIPS)

Sharing

- Shared physical pages through mappings
- This raises issues with the cache
  - Synonym problem: we will not address that here
The Memory Hierarchy

The BIG Picture

- Common principles apply at all levels of the memory hierarchy
  - Based on notions of caching
- At each level in the hierarchy
  - Block placement
  - Finding a block
  - Replacement on a miss
  - Write policy

Concluding Remarks

- Fast memories are small, large memories are slow
  - We really want fast, large memories 😎
  - Caching gives this illusion 😊
- Principle of locality
  - Programs use a small part of their memory space frequently
- Memory hierarchy
  - L1 cache ↔ L2 cache ↔ ... ↔ DRAM memory ↔ disk
- Memory system design is critical for multiprocessors
Study Guide

- Be able to trace through the page table and cache data structures on a memory reference (see sample problems)
- Understand how to allocate virtual pages to page frames to minimize conflicts in the cache
- Relationships between address translation, page size, and cache size.
  - For example, given a memory system design (page sizes, virtual and physical address spaces, cache parameters) understand the address breakdowns at different levels of the memory hierarchy
- Be able to map lines in a page to sets in the cache (identify the set from the address)

Study Guide

- Given a cache design and virtual address space and page size, define the pages (by their addresses) that may conflict in the cache
- Distinguish between a TLB miss, a data cache miss, and a page fault
<table>
<thead>
<tr>
<th>Glossary</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Page Table</td>
</tr>
<tr>
<td>• Page Table Entry (PTE)</td>
</tr>
<tr>
<td>• Page fault</td>
</tr>
<tr>
<td>• Physical address</td>
</tr>
<tr>
<td>• Physical page</td>
</tr>
<tr>
<td>• Physically tagged cache</td>
</tr>
<tr>
<td>• Translation lookaside buffer (TLB)</td>
</tr>
<tr>
<td>• Virtual address</td>
</tr>
<tr>
<td>• Virtual page</td>
</tr>
<tr>
<td>• Virtually tagged cache</td>
</tr>
</tbody>
</table>