Virtual Memory

Jinkyu Jeong (jinkyu@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
Memory system in 1950’s

lw R1, 4(R3) // R1 <- M[R3+4]

What is this address?
(called Effective Address)

• EDSAC, early 50’s
  – effective address = physical memory address

• Only one program ran at a time, with unrestricted access to entire machine (RAM + I/O devices)
  – Single-programmed execution environment
Dynamic Address Translation

• Motivation:
  – In the early machines, I/O operations were slow and each word transferred involved the CPU
  – Higher throughput if CPU and I/O of 2 or more programs were overlapped. Why?
    → multiprogramming

• Location independent programs:
  – Programming and storage management ease
    → need for a base register

• Protection:
  – Independent programs should not affect each other inadvertently
    → need for a bound register
Support for Location-Independent Programs (1)

- Base and bounds registers only visible/accessible when processor running in kernel (a.k.a supervisor) mode

Source: http://csg.csail.mit.edu/6.823/
Support for Location-Independent Programs (2)

- Separate areas for program and data

What is an advantage of this separation?

Permits sharing of program segments

Source: http://csg.csail.mit.edu/6.823/
Support for Location-Independent Programs (3)

- **What is the problem with segmentation?** **Fragmentation!**

As users come and go, the storage is “fragmented”.

-- Therefore, at some stage programs have to be moved around to compact the storage.


---

<table>
<thead>
<tr>
<th></th>
<th>OS Space</th>
<th>Users arrive</th>
<th>OS Space</th>
<th>Users leave</th>
</tr>
</thead>
<tbody>
<tr>
<td>user 1</td>
<td>16 K</td>
<td></td>
<td>user 1</td>
<td>16 K</td>
</tr>
<tr>
<td>user 2</td>
<td>24 K</td>
<td></td>
<td>user 2</td>
<td>24 K</td>
</tr>
<tr>
<td>free</td>
<td>24 K</td>
<td></td>
<td>user 4</td>
<td>16 K</td>
</tr>
<tr>
<td>free</td>
<td>24 K</td>
<td></td>
<td>user 3</td>
<td>32 K</td>
</tr>
<tr>
<td>free</td>
<td>24 K</td>
<td></td>
<td>user 5</td>
<td>24 K</td>
</tr>
<tr>
<td>user 1</td>
<td>16 K</td>
<td>free</td>
<td>user 4</td>
<td>16 K</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>user 4</td>
<td>8 K</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>user 3</td>
<td>8 K</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>free</td>
<td>32 K</td>
</tr>
<tr>
<td>user 5</td>
<td>24 K</td>
<td></td>
<td>free</td>
<td>24 K</td>
</tr>
</tbody>
</table>

---

**Note:** Diagrams illustrate the movement of users and their space usage over time, demonstrating the fragmentation and the need for compacting storage.
Outline

Textbook: P&H 5.4 (4th Ed.) / P&H 5.7 (5th Ed.)

• Virtual Memory: An Introduction
• Page Tables
• Translation Lookaside Buffer (TLB)
• TLB and Cache Interaction
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)
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
Page Tables (1)

• Stores placement information
  – Array of page table entries (PTEs), indexed by virtual page number
  – Page table register in CPU points to page table in physical memory

• If page is present in memory
  – 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
Page Tables (2)

• Translation Using a Page Table
Page Tables (3)

- Mapping Pages to Storage
Replacement and Writes

• 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
  – Block at once, not individual locations
  – Write through is impractical
  – Use write-back
  – Dirty bit in PTE set when page is written
Translation Lookaside Buffer (TLB) (I)

- Address translation would appear to require extra memory references
  - One to access the PTE
  - Then the actual memory access

- But access to page tables has good locality
  - So use a fast cache of PTEs within the CPU
  - Called a Translation Look-aside Buffer (TLB)
  - Typical: 16–512 PTEs, 0.5–1 cycle for hit, 10–100 cycles for miss, 0.01%–1% miss rate
  - Misses could be handled by hardware or software
Translation Lookaside Buffer (TLB) (2)

• Fast Translation Using a TLB
Translation Lookaside Buffer (TLB) (3)

• 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
Translation Lookaside Buffer (TLB) (4)

• TLB Miss Handler
  – TLB miss indicates
    • Page present, but PTE not in TLB
    • Page not preset
  – 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: Handling Page Faults

- Use faulting virtual address to find PTE
- Locate page on disk
- Choose page to replace
  - If dirty, write to disk first
- Read page into memory and update page table
- Make process runnable again
  - Restart from faulting instruction
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)
Review 1: Address Translation

Virtual Address

- TLB Lookup
  - TLB Lookup
    - miss
      - Page Table Walk
        - the page is
          - $\notin$ memory
            - Page Fault (OS loads page)
              - SEGFAULT
                - Where?
                  - Restart instruction
            - $\in$ memory
              - Update TLB
        - hit
      - Protection Check
        - denied
          - Physical Address (to cache)
            - permitted
          - update TLB
            - Protection Fault
              - SEGFAULT
                - where?
Review 2: Handling Page Fault

1. Reference
2. Trap
3. Page is on backing store
4. Bring in missing page
5. Reset page table
6. Restart instruction

load M

operating system

page table

free frame

physical memory
TLB and Cache Interaction (I)

- 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
TLB and Cache Interaction (2)

- Physically addressed caches (cont’d)
  - Allows multiple processes to have blocks in cache at the same time.
  - Allows multiple processes to share pages.
  - Address translation is on the critical path.

[Source: Lecture notes for ICE 3003 by Prof. Jin-Soo Kim @ SKKU]
TLB and Cache Interaction (3)

- Virtually addressed, virtually tagged caches
  - Homonym problem:
    - Each process has a different translation of the same virtual address.
  - Address synonyms or aliases problem.
    - Two different virtual addresses point to the same physical address.

[Source: Lecture notes for ICE 3003 by Prof. Jin-Soo Kim @ SKKU]
**TLB and Cache Interaction (4)**

- Virtually addressed, physically tagged caches
  - Use virtual address to parallel access to the TLB and cache.
  - TLB produces the PFN – which must match the physical tag of the accessed cache line for it to be a “hit”.

[Source: Lecture notes for ICE 3003 by Prof. Jin-Soo Kim @ SKKU]