Discussion 12

Virtual Memory

Aditya Balasubramanian

`aditbala [at] berkeley [dot] edu`

Announcements 📣

Agenda

- Virtual Memory

Virtual Memory

Physical Memory

- Limitations
  - Physical hardware space is limited!
  - Some physical memory are reserved for OS (e.g. I/O devices)
  - All programs share the same physical memory
    - How does a program know which memory is theirs?
    - How do we prevent a program from accessing other program’s memory or - even OS reserved memory?
- Solution - virtual memory!

Virtual Memory

- Solution
  - Forces programs to think that they have all the physical memory for themselves!
  - In reality, only has a small uniform chunk of physical memory - allocated per program
  - Done via software allocation and virtualization
- Terminology
  - Physical address space: actual part of memory (blackboxed from user - program)
  - Virtual address space: what user programs know about
  - Memory Management Unit: maps virtual -> physical
    - Isolated from user program (not accessible)
    - Smart swapping: when physical memory is full, put some stuff in disk

Terminology + formulas

- Page
  - A unit loaded from physical memory (similar to blocks in caching)
  - Same size of virtual & physical memory
- Offset
  - Index of word/byte in a page
  - # of bits used = log2(# of words/bytes in a page)
- Physical Page Number (PPN)
  - Index of a page in physical memory
  - # of bits used = log2(# of pages in physical memory)
- Virtual Page Number (VPN)
  - Index of a page in virtual memory

Page Tables

- Entries to the page table contains
  - Valid bit, permission bits, PPN
- Indexed by Virtual Page Number
- Stored in main memory
  - That means we access memory when we're trying to find mappings…
  - PTs also take up physical pages, sometimes multiple physical pages
  - Inefficient since most of the mappings won't get used all the time
    - Solution: multi-level page table

TLB (Translation Lookaside Buffer)

- Virtually a cache for page tables!
  - Fully associative + LRU replacement policy
- Usually very very small (hence not as costly if fully associative)
- Entries of TLB
  - VPN (used as the tag)
  - valid bit, permission bits, PPN
- Checked first upon receiving a virtual address
  - TLB Hit: no need to load PT
    - VPN match AND valid bit = 1
  - TLB Miss: look through PT
- Protection faults
  - Entry has correct valid bit but invalid permission bits
  - Segfault!

- Benefits: reduce main memory access!

Virtual Memory Workflow

1. Access given virtual memory
2. Breakdown virtual memory address
3. Look through TLB to find an entry with the given VPN. If TLB Hit, then use the found PPN and move on to step 6
4. Upon a TLB Miss, access the page table - index into the page table with the given VPN. If Page Hit, then use the found PPN and move on - to step 6
5. Upon a Page Fault, we need to load memory from disk
6. With the correct physical page number, reconstruct the physical - address by appending offset to PPN
7. Access physical memory

Multi-level Page Tables

- Inefficient to have to load a single-level page table since it's large
- Break it down so that we only load a couple of pages
- VPN1 indexes Level 1 Page Table
  - L1 PT entry: valid + permission bits, PPN of the L2 PT
- VPN2 indexes Level 2 Page Table
  - L2 PT entry: valid + permission bits, PPN of actual page
- # of L2 Page tables = # of PT entries for L1 PT
- Only tables corresponding to the page table entry will be loaded, so we don't need as many main memory accesses

Thank you!