TLB. Mac OS Memory Management

ECE595
Oct 23

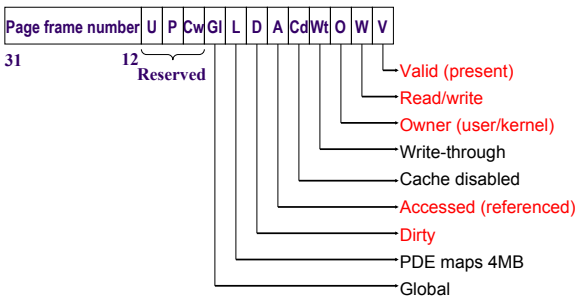
Y. Charlie Hu

[lec15] What is happening before and after malloc()?

- Before malloc()?
- After malloc()?
- Upon first access?

- How to capture the first write to a virtual page?
  - e.g. want to trap into page fault handler
    - Use valid bit
  - In handler, check if vpage is malloced.
    - If not, segmentation fault
    - Else allocate physical page

x86 Page Table Entry



[lec16] Virtual Memory

- Definition: *Virtual memory* permits a process to run with only some of its virtual address space loaded into physical memory

- Key idea: Virtual address space translated to either
  - Physical memory (small, fast) or
  - Disk (backing store), large but slow

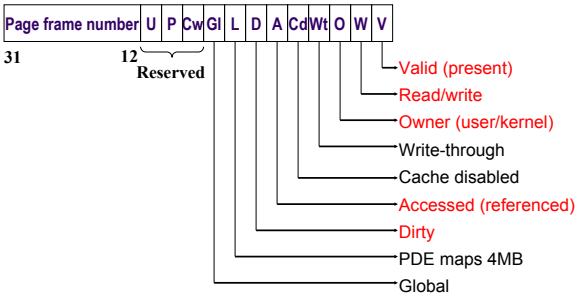
- [Deep thinking] What made above possible?

- Objective:
  - To produce the illusion of memory as big as necessary

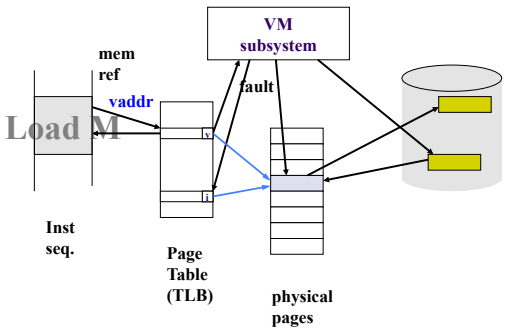
Demand Paging (paging with swapping)

- If not all of a program is loaded when running, what happens when referencing a byte not loaded yet?
- Hardware/software cooperate to make things work
  - Extend PTEs with an extra bit "present"
  - Any page not in main memory right now has the "present" bit cleared in its PTE
  - If "present" isn't set, a reference to the page results in a trap by the paging hardware, called *page fault*
  - What needs to happen when page fault occurs?

x86 Page Table Entry



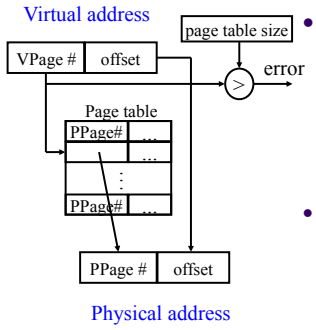
Page Fault Handling in Demand Paging



Today's topics

- TLB
- Mac OS Memory Management

[lec14] Paging implementation – how does it really work?

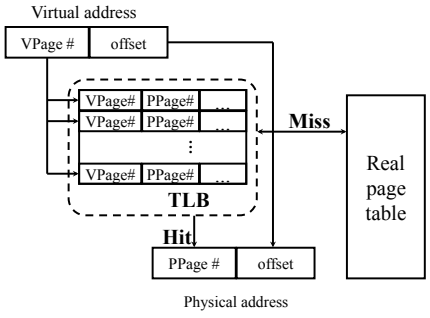


- Where to store page table?
- How to use MMU?
  - Even small page tables too large to load into MMU
  - Page tables kept in mem and MMU only has their base addresses
    - What does MMU have to do?
- Page size?
  - Small page -> big table
    - 32-bit with 4k pages
  - Large page ->small table but large internal fragmentation
    - e.g., data seg is 6 Kbytes

Performance problem with paging

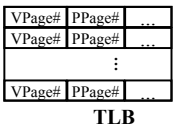
- How many extra memory references to access page tables?
  - One-level page table?
  - Two-level page table (midterm problem)?
- Solution: *reference locality!*
  - In a short period of time, a process is likely accessing only a few pages
  - Instead of storing only page table starting address in MMU,

Translation Look-aside Buffer (TLB)



TLB often fully set-associative → least conflict misses
Expensive → typically 64 – 1024 entries

Bits in a TLB Entry



- Common (necessary) bits
  - Virtual page number: match with the virtual address
  - PTE
- Optional (useful) bits
  - ASIDs -- Address-space identifiers (process tags)

Miss handling: Hardware-controlled TLB

- On a TLB hit, MMU checks the valid bit
  - If valid, perform address translation
  - If invalid (e.g. page not in memory), MMU generates a page fault
    - OS performs fault handling
    - Restart the faulting instruction
- On a TLB miss
  - MMU parses page table and loads PTE into TLB
    - Needs to replace if TLB is full
    - PT layout is fixed
  - Same as hit …

Miss handling: Software-controlled TLB

- On a TLB hit, MMU checks the valid bit
  - If valid, perform address translation
  - If invalid (e.g. page not in memory), MMU generates a page fault
    - If page not valid, OS performs page fault handling
    - Restart the faulting instruction
- On a TLB miss, HW raises exception, traps to the OS
  - OS parses page table and loads PTE into TLB
    - Needs to replace if TLB is full
    - PT layout is flexible
  - Same as in a hit…

Hardware vs. software controlled

- Hardware approach
  - Efficient – TLB misses handled by hardware
  - OS intervention is required only in case of page fault
  - Page structure prescribed by MMU hardware -- rigid
- Software approach
  - Less efficient -- TLB misses are handled by software
  - MMU hardware very simple, permitting larger, faster TLB
  - OS designer has complete flexibility in choice of MM data structure
    - e.g. 2-level page table, inverted page table)

Deep thinking

- Without TLB, how MMU finds PTE is fixed
- With TLB, it can be flexible, e.g. software-controlled is possible
- What enables this?

More TLB Issues

- Which TLB entry should be replaced?
  - Random
  - LRU
- What happens when changing a page table entry (e.g. because of swapping, change read/write permission)?
  - change the entry in memory
  - flush (eg. invalidate) the TLB entry
    - INGLPG on x86

What happens to TLB in a process context switch?

- During a process context switch, cached translations can not used by the next process
  - Invalidate all entries during a context switch
    - Lots of TLB misses afterwards
  - Tag each entry with an ASID
    - Add a HW register that contains the process id of the current executing process
    - TLB hits if an entry's process id matches that reg

Cache vs. TLB

- Similarities:
  - Both cache a part of the physical memory
- Differences:
  - Associatively
    - TLB is usually fully associative
    - Cache can be direct mapped
  - Consistency
    - TLB does not deal with consistency with memory
    - TLB can be controlled by software

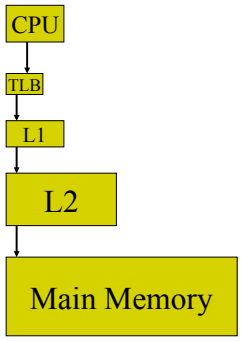
More on consistency Issues

- Snoopy cache protocols can maintain consistency with DRAM, even in the presence of DMA
- No hardware maintains consistency between DRAM and TLBs:
  - OS needs to flush related TLBs whenever changing a page table entry in memory
- On multiprocessors, when you modify a page table entry, you need to do "*TLB shoot-down*" to flush all related TLB entries on all processors

Memory Hierarchy Revisited

What does this imply about L1 addresses?

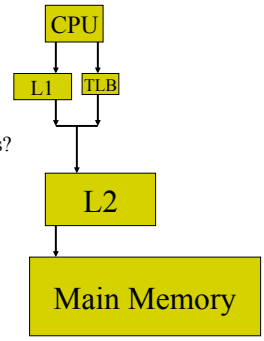
Where do we hope requests get satisfied?



Memory Hierarchy Re-Revisited

What does this imply about L1 addresses?

Any speed benefits?
Any drawbacks?



Today's topics

- TLB
- Mac OS Memory Management

Instances of extra level of indirection

- Dynamic memory reloction
  - Base and nound
  - Segmentation
  - 1-level paging
  - 2-level paging
  - Segmentation plus paging
- The MAC example

Classic example of indirection: MAC OS memory management

- Macintosh had 128 KB (~1985)
- Ran one app at a time
- Memory constraint was high priority

- Solution?
  - willing to trade off running time
  - "All computer science problems can be solved with an extra level of indirection."

Reading

- Chapter 8