Page Faults

Page fault:
• occurs when a page is not in memory
  • valid bit in its PTE is clear
• trap to the operating system to service the page fault
• page fault handler: the software that handles the page fault
  (next slide)

Demand paging: bring a page into memory the first time the CPU
references a location in it

---

Page Faults

What happens on a page fault (high-level view):
• choose a page frame to free (page replacement):
  • the algorithm approximates LRU replacement
  • reference bit is set on an access to the page
  • cleared every once in awhile
  • pick a page with a cleared reference bit
• if the dirty bit is set, write the replaced page to disk
• update its PTE (valid bit, dirty bit)
• read the faulting page from disk
• update its PTE
Page Faults

Disk overhead is large (milliseconds)
The implications:
  • want to reduce the page fault rate because servicing the page fault is expensive
  • mechanisms for maintaining a low page fault rate:
    • pages are at least 4KB to amortize the overhead of accessing it from disk & to reduce the page fault rate
    • fully associative mapping between pages & page frames to reduce page faults due to page frame conflicts
    • write-back disk update policy (disk writes take too long for write-through)
    • optimized page replacement algorithms to minimize page fault rate
  • have lots of time during a page fault because of the long disk latency
    • page fault can be handled in software
    • page replacement algorithms can be optimized (i.e., take time)
    • the program that incurred the page fault is descheduled & another program is scheduled to execute: called a **context switch**

Translation Lookaside Buffer

Translation lookaside buffer (TLB)
  • is a cache
  • contains the most recent virtual-to-physical translations
  • HW looks for the physical address in the TLB before checking the page table
    • if it’s there, avoid the memory reference to the page table
    • because of locality of reference, it probably will be there!

  • TLB configuration
    • usually fully-associative or large set-associative
    • 4-8 byte blocks
    • 32 - 128 entries (if fully associative), up to 4K if direct-mapped
    • can be instruction & data (today) or unified (more in the past)
    • write back
    • .5 - 1 cycle hit time, tens of cycles miss penalty
    • TLB misses handled in software or hardware or software with hardware assists
Using a TLB

(1) Access using the virtual page number. Why?

(2) If a hit,
   • concatenate the physical page number & the page offset bits, to form a physical address
   • set the reference bit
   • if writing, set the dirty bit

(3) If a miss,
   • get the physical address from the page tables
   • evict a TLB entry & update dirty/reference bits in the page tables
   • update the TLB with the new mapping

Using a TLB

TLB (physical) components:
• TLB entry (cache data)
  • contents of the PTE: physical page number, dirty bit, reference bit, protection bits
• TLB tags are process identifiers (PIDs) & virtual page numbers
  • PID prevents one process from accessing a TLB entry of another process
  • PID of the currently executing process is stored in a special register
  • TLB tag match: PID register & virtual address tag are compared to PID & virtual address in TLB tag
  • if a PID is not part of the tag match, how else can we prevent one process from accessing another process’s pages via the TLB?
• TLB state (valid, dirty bits)
Handling a Memory Reference

CPU emits a VA

TLB lookup

- hit
- miss

Cache lookup

- hit
- miss

Data to CPU

Cache miss

Mem access

Page table access

- valid
- invalid

Update PTE of replaced TLB entry

Insert new entry into TLB

Service the page fault

Insert new entry into TLB

Handling a Memory Reference

Special situations

- cannot hit in the TLB & have a page fault
  - TLB entry is invalidated when its page is paged out to disk
- cannot hit in the cache when there is a TLB miss and a page fault
  - blocks from the page are flushed from the cache when a paged is paged out to disk
- all other combinations are possible
  - see Figure 7.27
Pros & Cons of Paging

Advantages of paging:

• provides a simple memory location model to the programmer
  ⇒ users do not have to do manual overlays

• not all pages have to be in memory during execution
  (demand paging)
  ⇒ lower program start-up time

• program (virtual) space can be larger than the physical memory
  ⇒ allows larger programs or lower memory cost

• allows flexible page relocation; pages do not have to be contiguous (fully-associative)
  ⇒ low page fault rate

• allows co-location of programs in physical memory without (external) fragmentation
  ⇒ full utilization of memory

• allows programs to share pages
  ⇒ better use of memory

• provides protection of one program from another on a page basis

Pros & Cons of Paging

Disadvantages of paging:

• address translation via page tables takes time

• paging takes time