CS 3204
Operating Systems

Lecture 22

Godmar Back

Announcements

- Project 2 due Wed March 22
- Midterm Fri March 24
  - Can we start at 10:00am?
- Project 3 page table design document due Mon March 27
  - More specific information to follow
- Reminder: minimum requirement to pass the class is a working project 2
  - reread section on *Grading* in Syllabus
  - "working" means a >95% score as reported by "make grade"
  - project 2 is required for projects 3 & 4
- Reading assignments:
  - Stallings Chapter 7.1-7.4, 8.1-8.2

Virtual Memory

Page Tables & TLB
continued

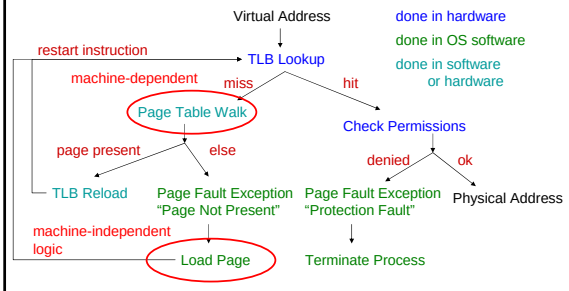
Page Tables as a Function

Trans (with paging):
{ Process Ids } × { Virtual Addresses } × { user, kernel } × $\wp$({ read, write, execute })

→ { Physical Addresses } ∪ { INVALID } ∪ { Some Location On Disk }

- Page Table: mathematical function "Trans"
- For each combination (process id, virtual_addr, mode, type of access) must decide
  - If access is permitted
  - If permitted:
    - if page is resident, use physical address
    - if page is non-resident, page table has information on how to get the page in memory
- CPU uses TLB for actual translation – page table feeds the TLB on a TLB miss

Address Translation & TLB



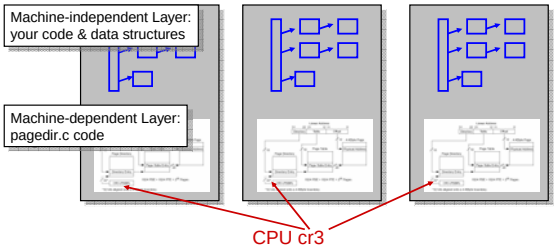
TLB Reloaded

- TLB small: typically only caches 64-2,048 entries
  - What happens on a miss? – must consult ("walk") page table – TLB Reload or Refill
- TLB Reload in software (MIPS)
  - Via TLB miss handlers – OS designer can pick any page table layout – page table is only read & written by OS
- TLB Reload in hardware (x86)
  - Hardware & software must agree on page table layout *inasmuch* as TLB miss handling is concerned – page table is read by CPU, written by OS
- Some architectures allow either (PPC)

Page Tables vs TLB Consistency

- No matter which method is used, OS must ensure that TLB & page tables are consistent
  - On multiprocessor, this may require "TLB shootdown"
- For software-reloaded TLB: relatively easy
  - TLB will only contain what OS handlers place into it
- For hardware-reloaded TLB: two choices
  - Use same data structures for page table walk & page loading (hardware designers reserved bits for OS's use in page table)
  - Use a layer on top (facilitates machine-independent implementation) – this is recommended approach for Pintos Project 3
    - In this case, must update actual page table (on x86: "page directory") that is consulted by MMU during page table walk
    - Code is already written for you in pagedir.c

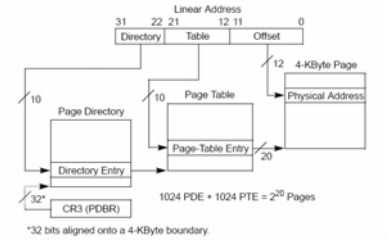
Hardware/Software Split in Pintos



Representing Page Tables

- Choice impacts speed of access vs size needed to store mapping information:
  - Simple arrays (PDP-11, VAX)
    - Fast, but space requirement not feasible for large, non-contiguous address spaces
  - Search trees (aka "hierarchical" or "multi-level" page tables)
  - Hash table

Example: x86 Address Translation



- Two-level page table
- Source: [IA32-v3] 3.7.1

Two-level Page Table

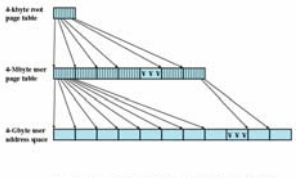
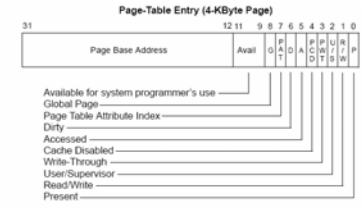


Figure 8.4 A Two-Level Hierarchical Page Table

- Q.: how many pages are needed in
  - Best case
  - Worst case? (what is the worst case?)

Example: x86 Page Table Entry

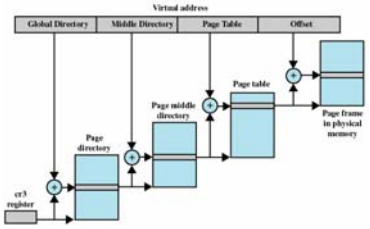


- Note: if bit 0 is 0 ("page not present") MMU will ignore bits 1-31 – OS can use those at will

Page Table Management on Linux

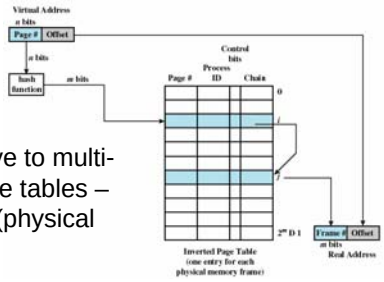
- Interesting history:
  - Linux was originally x86 only with 32bit physical addresses. Its page table matched the one used by x86 hardware
  - Since:
    - Linux has been ported to other architectures
    - x86 has grown to support 36bit physical addresses (PAE) – required 3-level page table
- Linux's now uses 4-level page table to support 64-bit architectures

Linux Page Tables (2)



- On x86 – hardware == software
  - On 32-bit (no PAE) middle directory disappears
- With four-level, "PUD" page upper directory is added (not shown)

Inverted Page Tables



- Alternative to multi-level page tables – size is O(physical memory)