15-213 Recitation 09

VM

Ben Blum (bblum@andrew.cmu.edu)

November 2, 2009

Outline

- Announcements / Questions
- Exam review - common misconceptions
- Virtual Memory
  - Theory
  - Single-level address translation
  - Multi-level address translation

Public Service Announcements

- Exam grades back
  - Drop deadline is today - talk to your advisor
- Malloclab checkpoint thursday
  - It's possible to "pass" but still be in trouble.
- Tshlab inkings back this week

Any questions (malloclab, perhaps)?

Returning from functions

- Return **address**: where you go when you are done
  - On the **stack** (at %ebp+4)
- Return **value**: the value that the caller sees you returning
  - In a **register** (namely, %eax)

Stack frames - saving %ebp

- %ebp is a callee-save register.
- After entering function
  - save %ebp on stack
  - assign new %ebp
  - allocate stack frame
- Before returning from function
  - deallocate stack frame
  - pop old %ebp from stack

Foreground vs background jobs

- The kernel doesn't care about "foreground" or "background"
  - abstraction provided by the shell
- Foreground jobs:
  - Shell waits - `sigsuspend` or `waitpid`
    - Wait on a *process*, not a *process group*!
  - Has control of the terminal (`tcsetpgrp`)

Why use VM?

- Process's private address space
- Can't see other processes' memory

How?

- Implemented in *hardware*
- When a memory access occurs:
  - CPU talks to MMU (memory management unit)
  - MMU does address translation
  - Virtual address converted to physical address
- `mov 0xdeadbeef,%eax` actually a different *physical* address

VM specifics:

- Page size 4KB (2^{12} bytes)
- Address length: 32 bits
  - 12 bits for page offset
  - $32 - 12 = 20$ addressing bits
- One-level page table
  - Located at `0x01000000` (arbitrary)
  - 4-byte PTEs
    - 4KB aligned, so lowest 12 bits always zero
    - Lowest 3 bits used for permission flags:
    - bit 0: mapping is present
    - bit 1: page is writable
    - bit 2: page accessible by user
  - How big overall?
    - 2^{20} indices, so 4MB

Simple example

- Virtual address `0xdeadbeef`
  - Physical Page Offset: `0xeef`
  - Page Table Index: `0xdeadb` (`1101 1110 1010 1101 1011`)

Simple example

- Virtual address `0xdeadbeef`
  - Physical Page Offset: `0xeef`
  - Page Table Index: `0xdeadb` (`1101 1110 1010 1101 1011`)
- Page Table Entry
  - Location: base + (size * index)
    - `11 0111 1010 1011 0110 1100` = `0x37ab6c`
    - Final: `0x0137ab6c`
  - Entry contents: `0x98765007` (arbitrary)
- Final physical address: `0x98765eef`

VM specifics:

- Page size 4KB (2^{12} bytes)
- Address length: 32 bits
  - 12 bits for page offset
  - $32 - 12 = 20$ addressing bits
- Two-level page directory+table
  - Directory located at `0x00010000` (arbitrary)
  - 4-byte PDEs, PTEs
    - 4KB aligned, so lowest 12 bits always zero
    - Permission bits for pagetables same as before
    - Page directory entries only use the "present" bit
  - How big overall?
    - 2^{10} indices (why?), so 4KB
- Protip: This is what x86 looks like

Advanced example

- Virtual address `0xdeadbeef`
  - Physical Page Offset: `0xeef`
  - Addressing bits: `0xdeadb` (`1101 1110 1010 1101 1011`)
    - **1st index:** `11 0111 1010`
    - 2nd index: `10 1101 1011`

Advanced example

- Virtual address `0xdeadbeef`
  - Physical Page Offset: `0xeef`
  - Addressing bits: `0xdeadb` (`1101 1110 1010 1101 1011`)
    - **1st index:** `11 0111 1010`
    - 2nd index: `10 1101 1011`
- Page directory entry
  - Location: base + (size * index)
    - `1101 1110 1000` = `0xde8`
    - Final: `0x00010de8`
  - Entry contents: `0x00011001` (arbitrary)

Advanced example

- Virtual address `0xdeadbeef`
  - Physical Page Offset: `0xeef`
  - Addressing bits: `0xdeadb` (`1101 1110 1010 1101 1011`)
    - 1st index: `11 0111 1010`
    - **2nd index:** `10 1101 1011`

Advanced example

- Virtual address `0xdeadbeef`
  - Physical Page Offset: `0xeef`
  - Addressing bits: `0xdeadb` (`1101 1110 1010 1101 1011`)
    - 1st index: `11 0111 1010`
    - **2nd index:** `10 1101 1011`
- Page table base = `0x00011000`
- Page table entry
  - Location: base + (size * index)
    - `1011 0110 1100` = `0xb6c`
    - Final: `0x00011b6c`
  - Entry contents: `0x98765007` (arbitrary)
- Final physical address: `0x98765eef`

Questions?