CSE 451 Section

Assignment 3

Virtual Memory

- Important mechanism, enables:
  - Isolation and protection
  - Virtualization: physical memory layout hidden
- OS sets up mapping: virtual -> physical address
  - Page table: translation structure
- CPU checks and translates on memory accesses
  - Translations cached by CPU for performance: TLB

Paging / Swapping

- Create illusion of more physical memory
  - (Still limited by size of backing store)
- Physical memory treated as cache for backing store
  - If we access not in cache, fetch from backing store
  - Might have to evict something else
- A number of design choices:
  - When do you write back dirty pages? Eagerly vs. lazily?
  - Which page is evicted?

MIPS virtual memory

- MIPS has software-loaded TLB
  - Page table lookup is implemented in software
  - TLB miss traps to kernel, kernel translates and adds to TLB
- 64 cache entries, fully associative:
  - Virtual page number
  - Physical page number
  - Valid and writable (dirty) bit
  - Adress space id (tag)
- This actually leads to some chicken-egg problems
  - Doing page table accesses in software will access memory

OS/161 memory layout

```
#define PAGE_SIZE

 * MIPS-I hardwired memory layout:
 *    0xc0000000 - 0xffffffff   kseg2
                (kernel, tlb-mapped)
 *    0xa0000000 - 0xbfffffff   kseg1
                (kernel, unmapped uncached)
 *    0x80000000 - 0x9fffffff   kseg0
                (kernel, unmapped, cached)
 *    0x00000000 - 0x7fffffff   kuseg
                (user, tlb-mapped)
```

Implementing translation in OS/161

- TLB miss → trap to kernel
  - `vm_fault(int faulttype,`
    `vaddr_t faultaddress)`
- Functions for manipulating TLB provided
  - See tlb_* functions in kern/arch/mips/include/tlb.h
- What should happen on a context switch?
- Eviction scheme?

Virtual memory with multi core

- TLBs are local to cores
  - Need to manually invalidate if other core changes mapping
- TLB shootdown

- OS/161 terminology
  - `ipi_tlbshootdown`: shoot down specified entries on specified CPU
  - `vm_tlbshootdown`: shoot down specified entries on this CPU
  - `vm_tlbshootdown_all`: shoot down all entries on this CPU
  - You need to implement `vm_tlbshootdown/_all`
    - Note: Shooting down all entries technically shoots down any specified entries

Core Map

- Mapping from physical pages to virtual pages
- Remember: core map must also be in physical memory!
  - Core map must be in core map
- How big should the core map be?
  - How many entries does the core map have?
- How do you reserve space for it?
- When should you reserve space for it?

Address Spaces

- High-level virtual memory abstraction
  - Page tables are built from this
- Consist of multiple disjoint segments
  - Generally larger than a page
  - Virtual address range, permissions
- See addrspace API

Swapping Implementation

- Where do you store swapped out pages?
- Multiple options: files, disk directly

- You can access the raw disk with vfs: “`lhd0raw:`"
  - Need to manage disk locations
  - How will you represent this information?
- Need to map pages to disk locations
  - Where will you keep this information?