Lab 3 Details
CSE 451 22wi
Admin

- Lab 3 design due 2/04 (tomorrow)
- Lab 3 due 2/18
Today’s Agenda

- More detail on vspace and vspace functions
- Some discussion questions on lab 3
- Hopefully, some time at the end for open questions
vregions vs Page Tables

- Both have virtual to physical address mappings
- **vspace.pgtbl**
  - Used by hardware to translate virtual addresses to physical addresses
  - **CR3** register holds the top level page table (i.e. `vspace.pgtbl`)
  - TLB caches virtual -> physical mappings
- **vspace.regions**
  - Portable *architecture independent* software representation of the address space
  - Used by kernel to track/update mappings without affecting hardware page table lookups
  - May be incomplete at times (e.g. mappings in `exec()`)
- How do we update the page table to reflect the vspace regions?
vspaceinvalidate(vs)

- “Transforms a vspace into the architecture dependent page table”
  - I.e. virtual mappings in `vs.regions` are reflected in `vs.pgtbl`
  - Git analogy: commit vspace changes to the page table
- Call when you’ve changed a mapping in `vs`.

Pop Quiz: When will you be calling `vspaceinvalidate` in Lab 3?
vspaceinstall\(p\)

- “Installs the page table into the page table register”
  - I.e. CR3 = vs.pgtbl
  - In x86, this flushes the TLB!
  - Git analogy: pushes your committed changes to the TLB/CR3
- If there were changes in the vspace, call after invalidating

Pop Quiz: When will you be calling vspaceinstall in Lab3? Can you ever get away without calling vspaceinstall?
Handling Page Faults in x86-64

- **CR2** register holds the faulting linear address (but since virtual paging is turned on, this is the virtual address)
  - How do you read or load a control register?
  - (look in `trap.c` in the default case)
- **tf->err** holds the exception error code
  - You can use this to determine the type of fault

More info: [https://wiki.osdev.org/Exceptions#Page_Fault](https://wiki.osdev.org/Exceptions#Page_Fault)
More on Error codes

- `rcr2()` returns address attempted to be accessed on page fault
- Last 3 bits of `tf->err`
  - B2 is set if fault occurred in user mode
  - B1 is set if fault occurred on a write
  - B0 is set if it was a page protection issue. This is not set if the page is not present

- What will the error code be if the page fault was from touching the stack region of memory?
- From touching a copy-on-write page?
Copy-on-write Fork FAQ

- How do we keep track of physical pages and refcounts?
  - Coremap!

- What vspace functions need to be modified to support COW fork, and how?
  - vspacecopy()

- What do the fields of a page (struct `vpage_info`) need to be after a copy-on-write fork?
  - fields to consider: used, ppn, present, writeable

- What happens to a page that is already read-only before COW fork?
More COW

- What needs to be changed in the `core_map_entry` to support COW fork?
  - Ref count, (and a lock for the core map)

- Can the kernel cause a copy-on-write page fault?
  - Sure! E.g. accessing the user buffer during a `read()` system call

- Synchronization in modifying the `vspace` in page fault in COW fork?
  - Not needed -- current process has exclusive access to its own vspace (no multithreading)
  - However, the `ref count` on the physical page could be concurrently modified

- What can happen if a copy-on-write fork is not synchronized?
Helper Macros and Functions

- **P2V**: physical addr to virtual addr
- **V2P**: virtual addr to physical addr
- **PGNUM**: physical addr to page number
- **va2vpage_info**: virtual addr to vpi_info
Any questions?