

## Lab 3 Overview

Don Porter  
CS372H - Spring 2007

## User Environments

- Like a process in Unix
  - Thread
  - Address Space
- But different
  - Shared kernel stack - more on this later!
- Environment state is stored in a struct env

## struct env

```
struct Env {
    struct Trapframe env_tf;    // Saved registers
    LIST_ENTRY(Env) env_link;  // Free list link pointers
    u_int env_id;              // Unique environment identifier
    u_int env_parent_id;       // env_id of this env's parent
    u_int env_status;          // Status of the environment

    // Address space
    Pde *env_pgdir;           // Kernel virtual address of page dir
    u_int env_cr3;            // Physical address of page dir
};
```

## UENVS

- env structs are exported to user code read-only in UENVS
  - Thus, environments can learn about themselves and others
  - Like a primitive /proc

## ... ELF parsing

- We don't have a filesystem, so program binaries are embedded in the kernel
- `load_icode()` gets a pointer to the appropriate binary
- Need to allocate pages and copy executable
  - How do you copy?
- Also allocate blank pages for BSS and other sections

## ... ELF Parsing, cont. (inc/elf.h)

- Binary begins with a struct `elf`
- `elf` contains an array of struct `Proghdr`'s at `elf->e_phoff`
- `Proghdr` contains:
  - `va`
  - `Memsz` - size of section
  - `filesize` - size of actual data to populate the section
  - `p_phoff` - offset into binary of the section to copy
  - `p_type` - only copy if it is `ELF_PROG_LOAD`
- The starting instruction is in `e_entry`
- For extra help, check out `readelf(1)` and `elf(5)`

## ... Running an environment

- As easy as loading the right registers
  - Hope page dir is right!
- And executing the 'iret' instruction
- Check out `env_pop_tf`
  - Hint: Helpful for `trapentry.S`

## ... Interrupts, Exceptions, & Traps

- Hardware events that are handled by code at pre-defined locations
  - Interrupts - asynchronous (device)
  - Exceptions - synchronous (code exception)
  - Trap - System call
    - Implemented via interrupts (`int` instruction)
- A code location is specified for each type in the Interrupt Descriptor Table

## Interrupt Descriptor Table

- Use SETGATE macro to create gate descriptors for each interrupt
  - The skeleton code does the rest
- Descriptor must go to assembly code.
  - Why?
  - How do we get those code labels in C?
- Can all descriptors go to same code?
  - Why not?
  - Can we get close?

## trap()

- In trapentry.S, we can push the error code and trap number on the stack and then call a common routine - trap().
- trap\_dispatch() - where you will have a large if/else statement and plug in handlers
  - Once you get this far, you are on the downhill slope!