

## Lab 4 Introduction

Don Porter  
CS372H - Spring 2007

## Part A: Round Robin Sched

- Simple Scheme:
  - Process calls yield syscall
  - Next process in `envs[]` runs until yield or exit
  - When all processes end, go to monitor
  - Fairly straightforward

## fork()

- Typical usage in Unix system:

```
int pid = fork();
if(pid == 0){ // I am child
    // child code
    exit 0;
} else if(pid < 0){
    // pid == error code
    // something bad happened
} else { // I am parent
    // pid = actual child pid
}
// More parent code
```

## sys\_exofork()

- Similar usage to fork, but minimal function
  - Doesn't map anything below kernel space
  - Not runnable
- Build fork on top of this and other sys calls
  - `env_set_status()` - set status to runnable
  - `env_page_alloc`, `env_page_map`, `env_page_unmap` - self-explanatory

## Env. Modification Policy

- Not just anyone can muck with a process's virtual address space
  - Why not?
- JOS policy - only environment itself and immediate parent can modify address space

## Part B: COW Fork

- Copy-on-write
  - Standard practice in modern OSes
  - Child begins with identical copy of parent's address space
  - Writes by parent or child cause a page fault
    - handler makes unique copies for each
  - Why is this popular?
  - How might you implement it?

## COW Implementation

- Disable all write bits in the page tables
- Replace with COW bits
  - Leave Read-only pages read-only
  - COW bits come out of user-defined range - i.e. not part of x86 standard
- On a page fault, copy page, set W bit and remove COW bit

## But why do it in the kernel...

- When can you do the work in user mode?
- Part B includes code to dispatch page faults to user code
  - After all, we already have syscalls to alloc, map and unmap pages
- Same idea as regular trap handling, except kernel writes a user trapframe instead of CPU

### ...

## Nested User Exceptions

- The page fault handler can page fault!
- This is tricky, but you can do it!

### ...

## Part C: Preemptive Multitasking

- Rather than just switching on a yield(), change processes on a timer interrupt
  - Must set up irq handler for the timer
  - Use IF flag in EFLAGS register to enable interrupts in user space
    - Don't take timer interrupts in the kernel
      - Why not?

### ...

## Part C: IPC

- Take a look at System V IPC for background
  - Mutexes, pipes, shared mem, etc.
- Despite higher level abstractions, it is just clever manipulation of virtual memory mapping
  - There is no magic!

### ...

## IPC Implementation

- Other than page mapping tools you already know and love...
- Receiver must block until it has a message
  - Sender sets receiver back to runnable
  - Sender must be sure there is room in send buffer
    - Receiver may not want any messages
    - Message buffer can be full
      - Buffer overflow anyone?
- Can also send entire pages

