Process Creation

```
now let's talk about creating first process and its address space

Q: you'd expect there to be an array or something of pointers
   to the process's user memory. where is it? how does xv6
   know what memory a process is using?

ordinarily p->pgdir and phys mem contents created by fork()
  we will fake them for first process

ordinarily p->tf and p->context created by syscall/switch
  we will fake them for first process

main calls userinit

userinit sheet 22
  only called for first process
    other processes created by fork
  mimics fork+exec
    create a normal-looking process
    ordinary scheduler will run it
  needs to fill in all struct proc entries

allocproc sheet 22
  used by both fork and userinit
  kernel stack setup:
    trapframe w/ "saved user registers"
      for us, initial user registers
      eax, eip, esp, &c
    trapret !
    context w/ "saved kernel thread registers"
      for us, initial kernel thread registers
      eip
    assumes that execution will resume in a special
       assembly function called swtch (sheet 27) at line 2722,
       where there is code to pop the registers
       from stack and execute the return instruction
  Q: where will new kernel thread start executing?
  doesn't set up trapframe b/c ordinarily copied
    from parent by fork, which calls allocproc
  but fork and userinit both always start thread in forkret

trapframe sheet 06
context sheet 20

kernel stack diagram:
  top ->
                esp, ss
                eip, cs
                ...
                gs fs es ds
  p->tf ->      edi & 7 other registers
                ---
                trapret
                ---
                eip = forkret
                ebp
                ebx
                esi
  p->context -> edi
                ---
  p->kstack ->  ...

Q: any guesses why there are *two* saved EIPs?

back to userinit
  we know setupkvm -- only fills in kernel mappings
  this is a new page table for the new process
    not using it yet, will switch when new kernel thread starts
  call inituvm w/ ptr to new process's user instructions

initcode.S sheet 77 : becomes _binary_initcode_start
  user program
  exec("/init", args)

init.c sheet 78 :
  initializes fds 0, 1, 2
  forks shell and waits for it to exit.
  if shell exits, init exits too.
```

```
inituvm sheet 18
  we know kalloc and mappages(pgdir, va, sz, pa)
  initcode is tiny, fits in one page
  diagram: new mapping

Q: new page is mapped at va=0; could inituvm call memmove(0, init, sz)?

back to userinit sheet 22
  tf->esp -- user stack at top of page
  tf->eip=0 -- first instruction at bottom of page

main calls scheduler() sheet 12

scheduler sheet 24
  no longer initialization: kernel now fully running
  whenever process gives up CPU -> scheduler
  so kernel runs scheduler a lot
  look for a process that wants to run, run it
  p->state: SLEEPING, RUNNABLE, RUNNING
  scheduler looks for RUNNABLE

switchuvm sheet 17
  tell h/w to use p->stack if re-enters kernel
    sys call or interrupt
  load %cr3

let's watch switch to new process's page table:
  (gdb) break switchuvm
  (gdb) x/5i 0
  0x0:    Cannot access memory at address 0x0
  next past load %cr3
  (gdb) x/5i 0
  same as initcode sheet 75
  but we are still in the kernel, in scheduler

back to scheduler sheet 24
  mark RUNNING so no other CPU runs it
  now switch to new process's kernel stack, registers, EIP
  swtch(place to save current ESP, previously saved ESP to switch to)

let's watch:
  (gdb) break swtch
  si until esp switch...
  (gdb) x/6x $esp
  si past esp switch
  (gdb) x/6x $esp
  after: 4 regs, forkret, trapret

step into forkret sheet 24
  just returns
  allocproc set up stack to have it return to trapret
  watch out:
    release and initlog cause interrupts
    so hack source to set first=0 and pushcli
    si for iret &c -- si leaves interrupts off
  next into trapret

look at trapframe sheet 06
  (gdb) x/19x $esp
  0x0 0x23 are eip:cs
  0x1000 0x2b are esp:ss

trapret sheet 29
  pops trapret registers from stack, mostly zero
  popal pops 8 general-purpose registers
  iret pops ESP, EIP, clears supervisor flag
  x/5x $esp
  now we are executing at address 0x0 in initcode sheet 75

what does initcode do?
  traps back into the kernel to make exec() system call
```