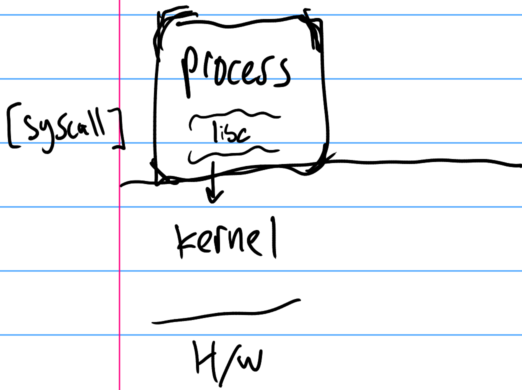
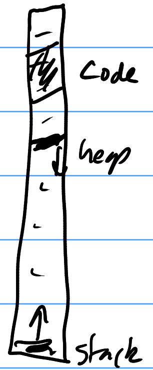
CS 3214 lecture #2 "Processes"



Process: abstraction of a running program

> - logical flow of controls through instruction
>   - "thread"
>   - ≥1 thread
>
> address space { stack, heap, code }
>
> write(1, "hello world", 12);
> (1 → stdout)



abstract view of resources

Why do we need it?

- originally: no OS
- mainframes: batch jobs
  - library I/O device code seems special
  - hlt
- minicomputers: digital PDP

UNIX {
- multiprogramming: switch programs if I/O
- time-sharing: switch programs so all make progress

OS dark ages — PC-era — DOS – no memory protection; macOS – only cooperative sched

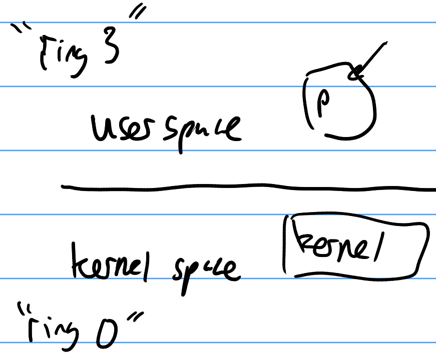
---

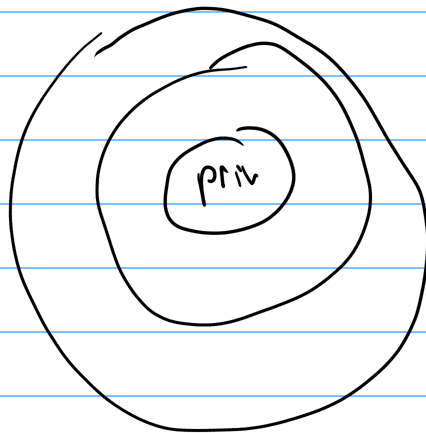
Now: UNIX-ideas are in all OS's

- Windows NT
- OSX
- Linux

protection: privileged instructions
- hlt
- I/O instruction
- change security policies
- access to entire filesystem

Dual Mode Operation   CPU — user
                           — supervisor





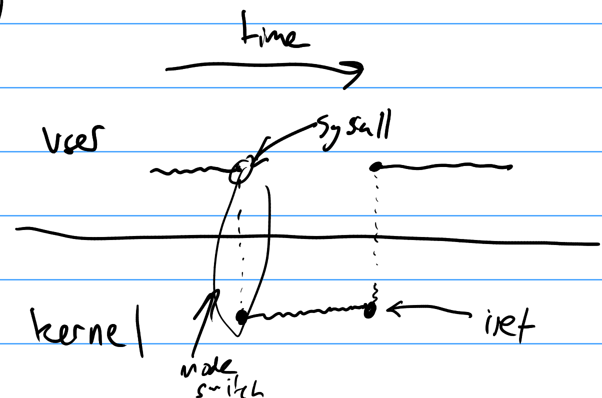
when does user → kernel
- system call
- switching processes] timer interrupt
                         device interrupt
- sync/internal interrupt
      divide by 0 (trap or exception)

async/external signal

trap handler
interrupt handler

when does kernel → user
  kernel issues iret (privileged)

  where do you return to?
  - instruction you left on
    - retry (e.g. page fault)
    - just after (e.g. syscall)



Containing the Omnipotent Root ← FreeBSD Jails

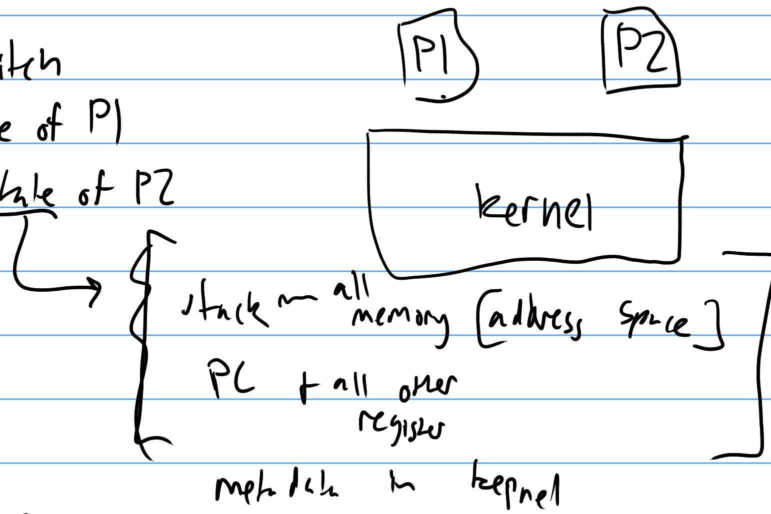
eBPF

CAP_NET_ADMIN, CAP_...

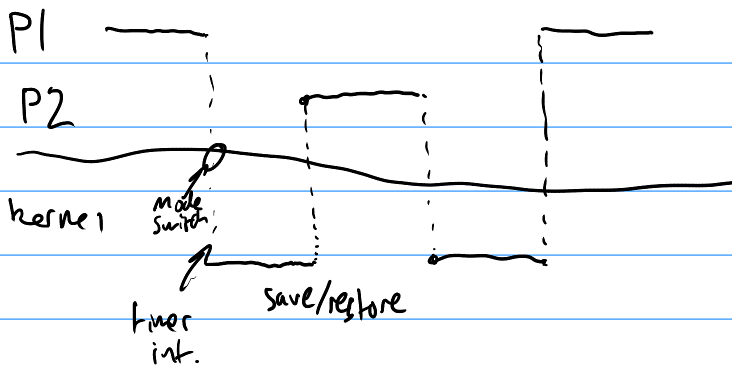
Attestation

Context switch

- save state of P1
- restore state of P2



→ time



[Limited Direct Execution] → insts run on CPU

- Limited → Can't run priv. inst.
- → can't take control forever