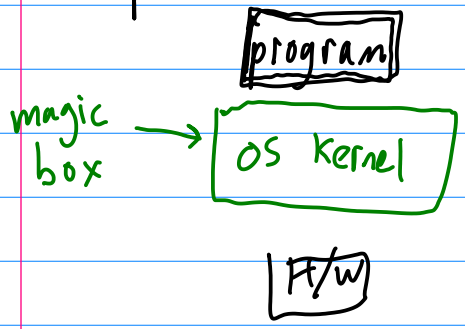
CS 3214 lecture #2: "the process"

Recap



magic tricks
- our own CPU
- our own memory 0x4044a
- abstracting device

System call interface
- syscall
  - any language (C, Java, Python, Javascript....

"Process"

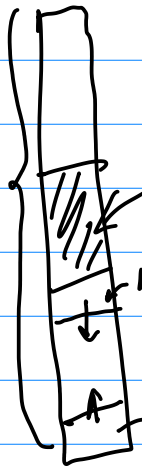
- abstraction of a running program

source code : list of instruction

→ logical flow of control through instruction (1 or more threads)

→ private address space

→ abstracted resources (fd's)



Why do we need processes?

History

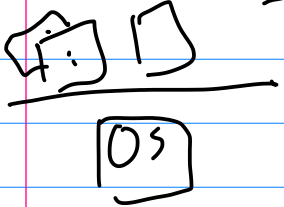
OS was a library

mainframes batch processing
one job at a time

OS is special, special privileges
"protection"

minicomputers - digital PDP (fewer people per computer)

→ multiprogramming (switch between user jobs if one is blocked on I/O)



UNIX → time sharing (switch between even non-blocked jobs so ALL make progress)

history

UNIX

dark ages

PC era

DOS — no memory protection

MacOS — only cooperative scheduling

now UNIX is back

→ Windows NT

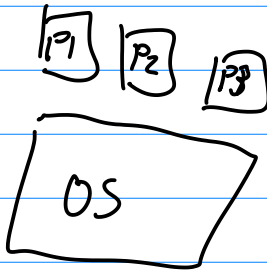
→ mac OS X IS UNIX

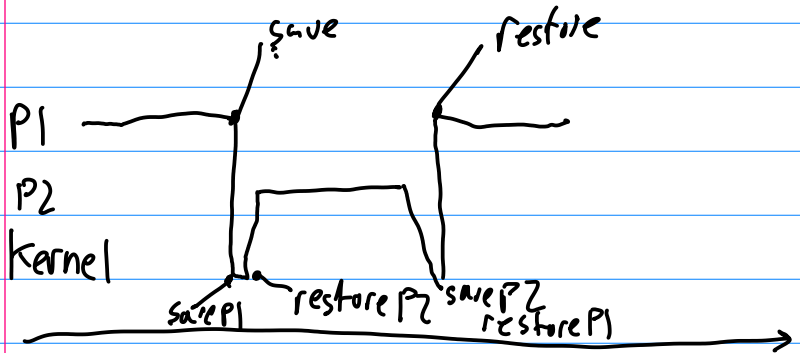
→ Linux is UNIX

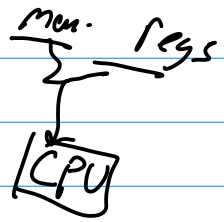
servers google, fb, ... android

UNIX : multiprogramming

time-sharing







"Context" switch

- save / restore P1's context

code
heap
stack
} mem "address space"

regs sp pc

Problems

- Protection → dual-mode operation
- how to get control back → interrupts (timer interrupt)

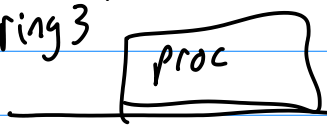
Dual-mode operation

- CPU bit "mode"
- rings, protection levels

PL 0
EL

x86 "ring 3"

x86 "ring 0"



Kernel

(userspace)

user

supervisor

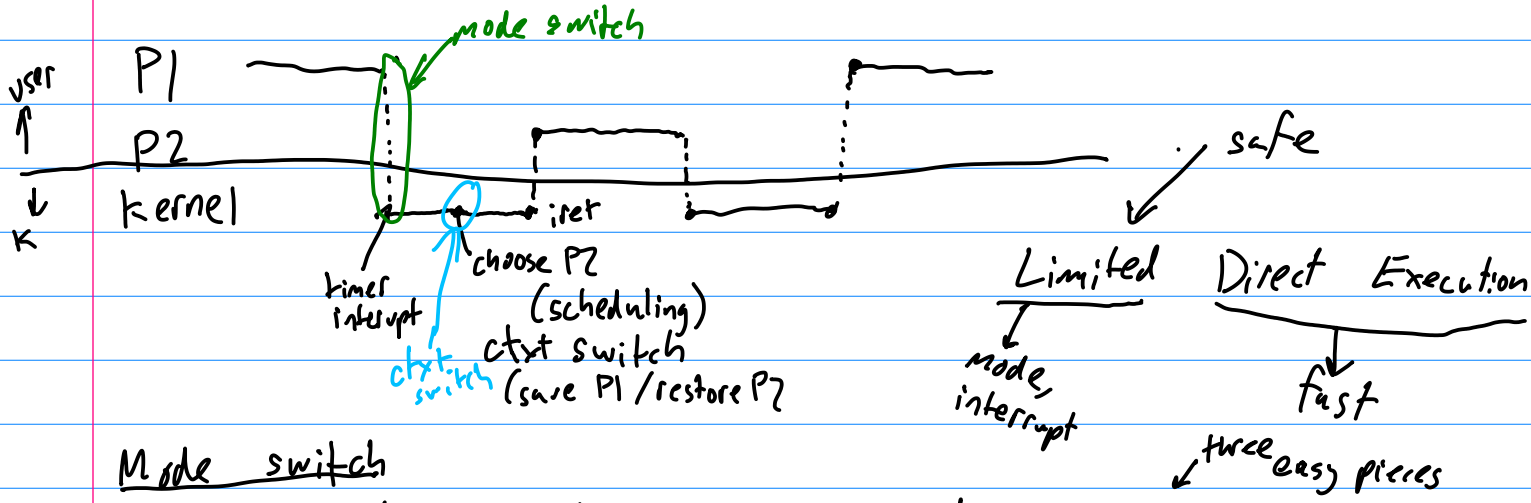
(kernel space)

Mode switch

User → kernel
- system calls (syscall)
- interrupts
  - internal (int 80, trap/exception)
  - external (times, I/O device, kbd, mouse...

kernel → User
- return from kernel (iret)

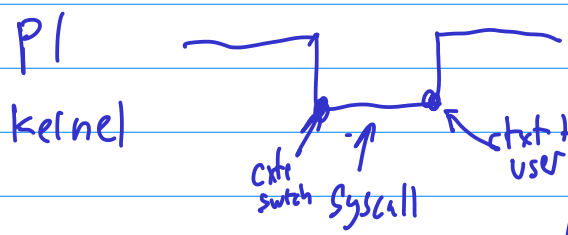
IDT



three easy pieces

book OSTEP
Remzi Andrea Arpaci-Dusseau

Mode switch
- ctxt switch w/o mode
- mode switch w/o ctxt switch



KPTI
kernel page table isolation
super expensive!

Meltdown 2018