# CS 537 : Day 2

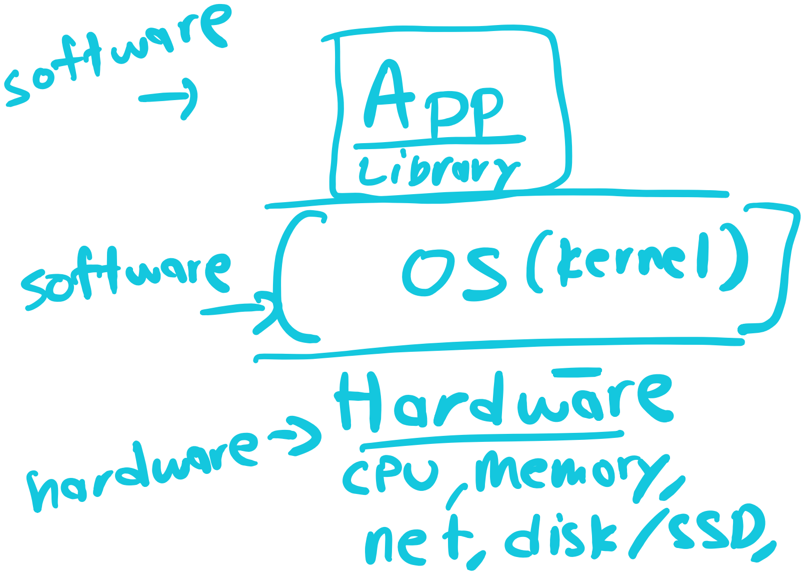
Operating Systems

virtualization ←

Concurrency

Persistence

easy to use
→ correctly
→ efficiently



## Virtualization

fact: physical limit in machines

⇒ 1, few CPUs, limited amount of DRAM

build an illusion

=> as many CPUs as needed

for each running program,
=> a large memory
(private)

Virtualization:

CPU

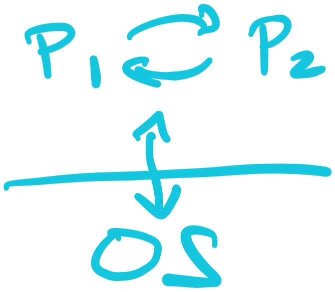
Memory

Goals:

-> Efficiency

-> Security
isolation

running program or
"process"

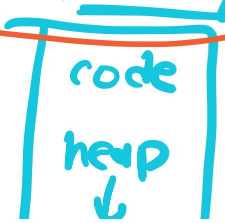


## Abstraction : Process

what changes when a
program runs?

1) -> registers
(PC, general
purpose)

2) -> memory ("address space")



3)

→ 3) I/O (input, output)
(e.g., open file descriptors)

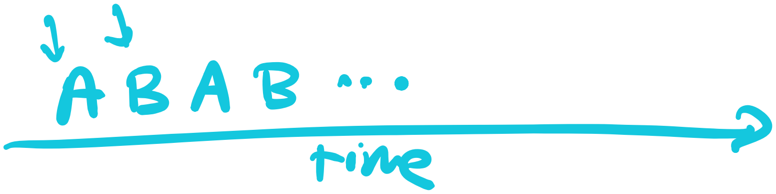
## CPU: how to virtualize?

⇒ run N processes "at once" even though we have M CPUs

(N > M)

general idea:

1 CPU, 2 processes A, B



"time sharing"

→ Mechanisms: low-level how

→ Policies: which process to run?

First attempt: Direct Execution

Boot time: (start up)

⇒ OS is the first prog. to run

set up:

-> free list (to track memory)

-> process list
(etc.)

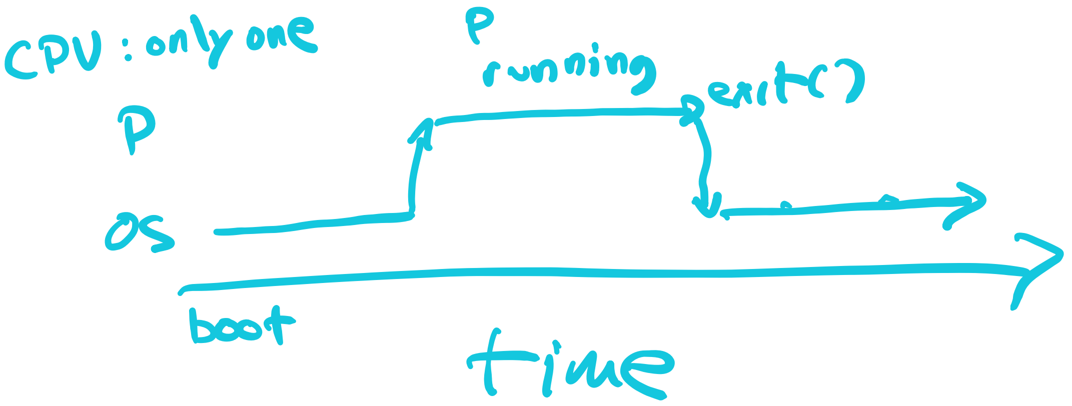
want: run one program

OS
=> alloc entry on process list

=> alloc some memory

=> load the program -> memory
(code, static data)
-> disk -> memory

=> jump to main of program + start running



observations:

only one thing running

@ a time

while(1);

[reboot]

Problems:

1) what if P wants to do something restricted? (e.g., issue a read to disk)

2) What if OS wants to stop $P_A$, run $P_B$?

3) what if P does something that is "slow"? (disk I/O, net I/O)

Class

web page ← Canvas

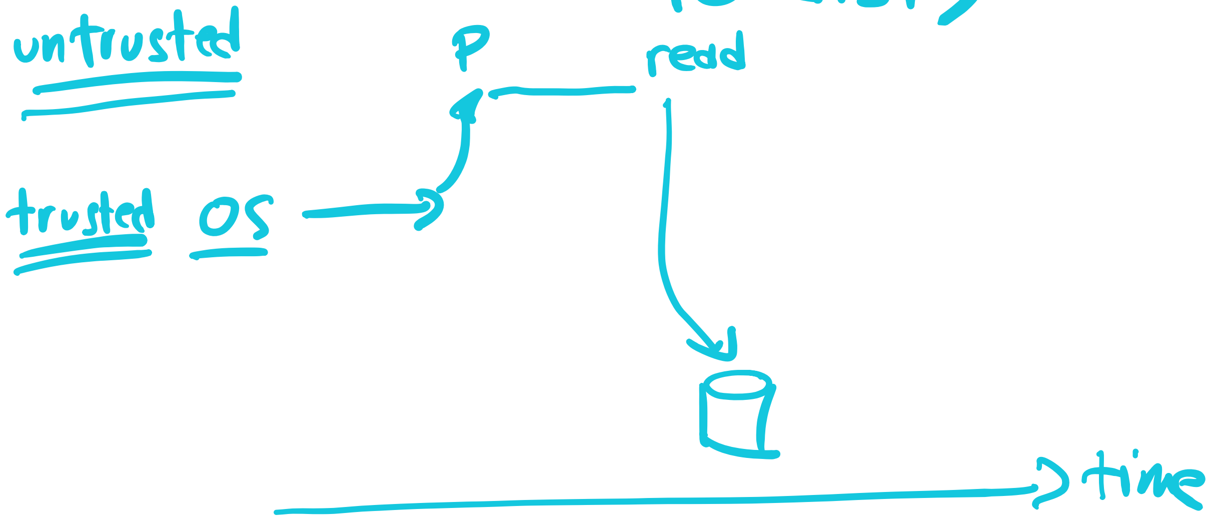
↓

1) Piazza → Discord

2) P1a due <u>Monday</u>

3) weekly <u>Canvas quiz</u>

1) what if P wants to
do something restricted?
(e.g., issue a read
to disk)



Hardware support:

1) modes of operation (per CPU)

"user mode" (applications)
restricted, non-privileged

"kernel mode" (OS)

unrestricted, privileged

(per CPU) bit: 0 or 1 => which mode

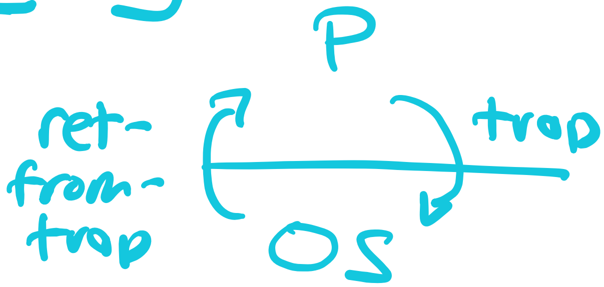
2) transition between modes

=> instructions:

user mode -> 1) enter kernel mode from user mode: trap

2) kernel -> user: return-from-trap

kernel mode -> [ 3) set up trap handlers ]



Time line:

OS @ boot

=> set up trap handlers

(+ other stuff)

modern OS:

Proc @ run time

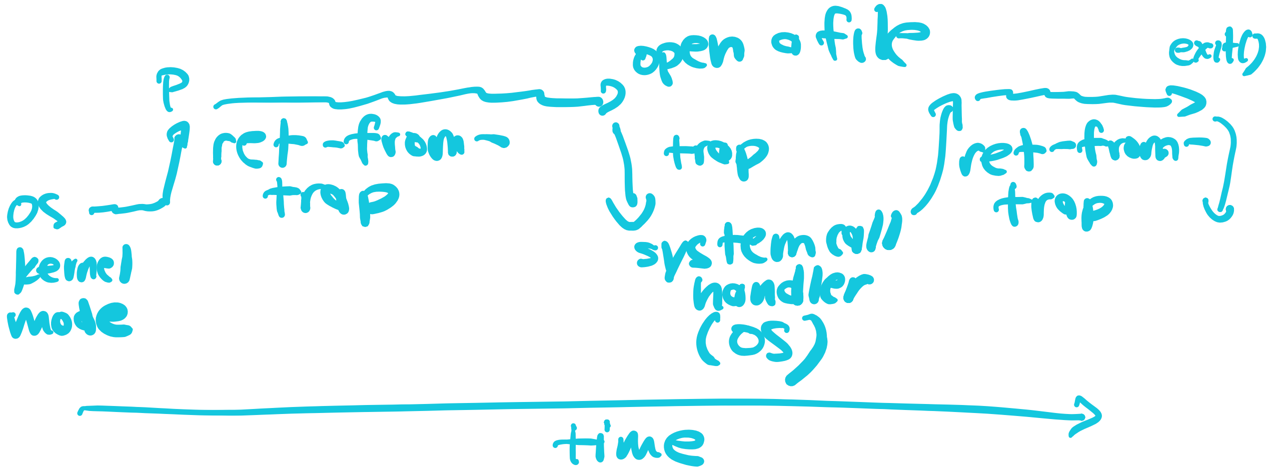
~200 sys calls

## system calls:

open

read → assembly: calls trap

close



⇒

## Limited Direct Execution