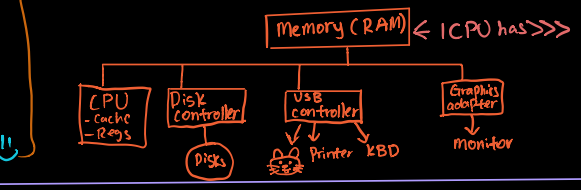
Week 1

What is an OS? It's just.... a program It's a government

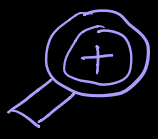
↳ Acts as an intermediary between users & hardware

Goals

- Resource allocator & coordinator ← Pg 2
  - ↳ controls hardware & I/O requests
  - ↳ manage conflict requests
  - ↳ manage interrupts.
- Controls program execution
  - ↳ storage hierachy manager
  - ↳ Process manager
  - ↳ security: ✗ illegal access to hardware !!
- Executes user programs
- Make computer system convenient to use
- Use computer hardware efficiently



- Manage Process
- manage resources
- Inter process communication



The kernel! it's just running all the time!

- access!
- memory hierachy + PA space
- 6 roles → interrupt user prog
  - scheduler
  - I/O req manager
  - manage user prog locations!

→ it's a Program.

Eg. Linux-kernel
Ubuntu - OS

It has.... simplified into 3.

- Special Privileges → like an administrator of the website.
- system calls let user programs access kernel services
- Dual modes
  - ↳ kernel: full access to hardware
  - ↳ User: Run on VM.

(sys calls)

memory hierachy
↳ Speed fast

How does the OS Run? Think of how the macbook starts.

Computer ON → BIOS Scan. → BIOS loads MBR → MBR loads boot loader → OS
- ▷ piece of code to Scan
- ▷ points to entry codes of OS

Computer system I/O operation

Run code/ machine instructions.

CPU ⇄ I/O device Controllers (hardware) → Reg → instructions read by device driver Program @ CPU

(inform thru interrupt)

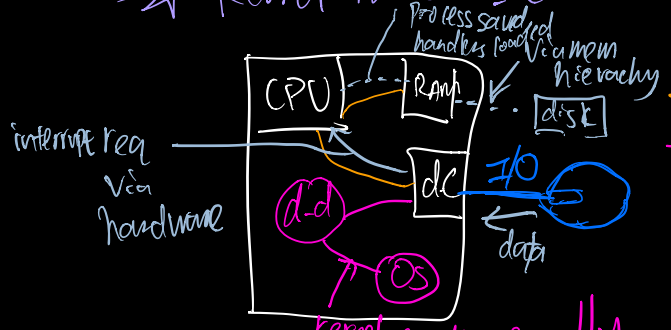
↓ talks to

device drivers (software) ↻ Local buffer → Instructions fetched by CPU & loaded into RAM

- ▷ key thing is to communicate/ understand the device type.

How so? software interface to hardware.

★ Kernel mode! (most of them). Why? switch to kernel mode needs to ACCESS serial ports.



Kernel mode usually.

Interrupts

- hardware → kbd/ mouse etc → CPU transfer to interrupt handler in kernel mode
  - ↓ save reg states
- software → trap.

- IVC — addr of all svc routines
- Save addr of interrupted inst
- Incoming interrupts disabled → prev lost interrupt/ reentrancy problems.
- Trap: user code error / request → sys call → kernel mode.

Qn: How can we determine which type of interrupt occured?

a. Polling — got interrupt? → action: accept address into interrupt vector
b. vector — where is the interrupt?

Async I/O handling (Refer to diagram above).

☆ System Calls !

↳ Eg. mkdir, cd, etc...

→ a. Traps → CPU go to special handler calls on interrupt service ratio to process in kernel mode.

↓ trap handler

↓ certain user states saved.

b. trap handler

c. certain user states are saved.

☆ Timed interrupts (hard here ⏲)

↓

counter == 0 {
ctrl to interrupt handler
state-save → scheduler }

☆ Exceptions — CPU checks for prog raised → event-vector tbl.

☆ memory management → Caching algo → keep track
→ decide to move in & out of memory

↓ updated when Page fault occurs.

Caching algo ↓ managed by hardware ↓ Swap space migration ↓ allocate + deallocate.

fetch new data from main memory

☆ $EAT = x(\tau) + (1-x)\varepsilon$

τ → access cache/RAM or disc

x → hit ratio, τ → hit, ε → miss

Multiprogramming — ↑ CPU utilization → how? organize jobs

- single users cannot keep CPU & I/O busy at all times
- Kernel orgs jobs efficiently
- Virtual memory — execution of processes not completely in memory

CPU one to execute.

→ time sharing — context switch → rapidly so Users still see system as interactive & seemingly capable to run multiple processes.

direct comm btw user & system

↓ how?

stores state of process.

Process vs Program

- Prog in execution
- Has context
- active entity — Passive entity
- terminates, resources are freed by kernel
- resides on disk
- doesn't take up resources

A CPU achieves concurrency by multiplexing

multiplexing

I/O handlers → processes → no context → handle certain events

☆ Process manager → scheduler subroutine → what should wait for what?

↓

Process table

☆ multiprocessor system

↓

asymm

↓

master-slave

advantages:

- ↑ throughput
- Economy of scale
- ↑ reliability

☆ clustered system — SAN → HPC

↳ Asymm

↳ Symm

!! :

- carefully control I/O
- Ensure load balancing
- cache coherency