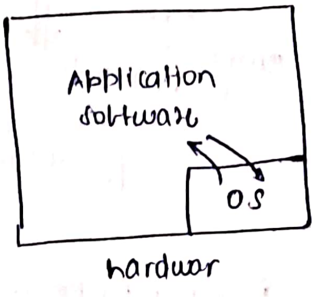
Operating System

linux have 15 million line of base code

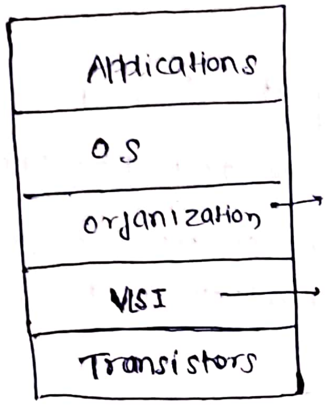
laptop → 4 - 8 cores

server compute → 80-100 cores

operating system is collection of libraries that allowing hardware to run

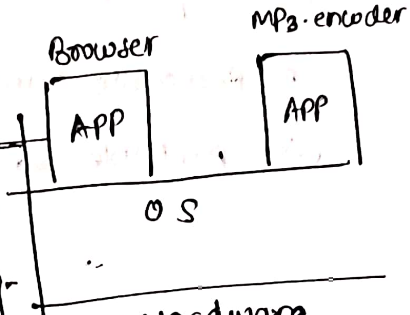


The layer in Systems



organization → General hardware we see eg → mouse, cpu, monitor

VLSI → very large scale integrating eg → flipflops, Gates, multiplexures, register etc



Both are interacting with operating system and OS is interacting with hardware

Hardware

① more security

② Both develober Browser & MP3.encoder are not dependendent on each other.

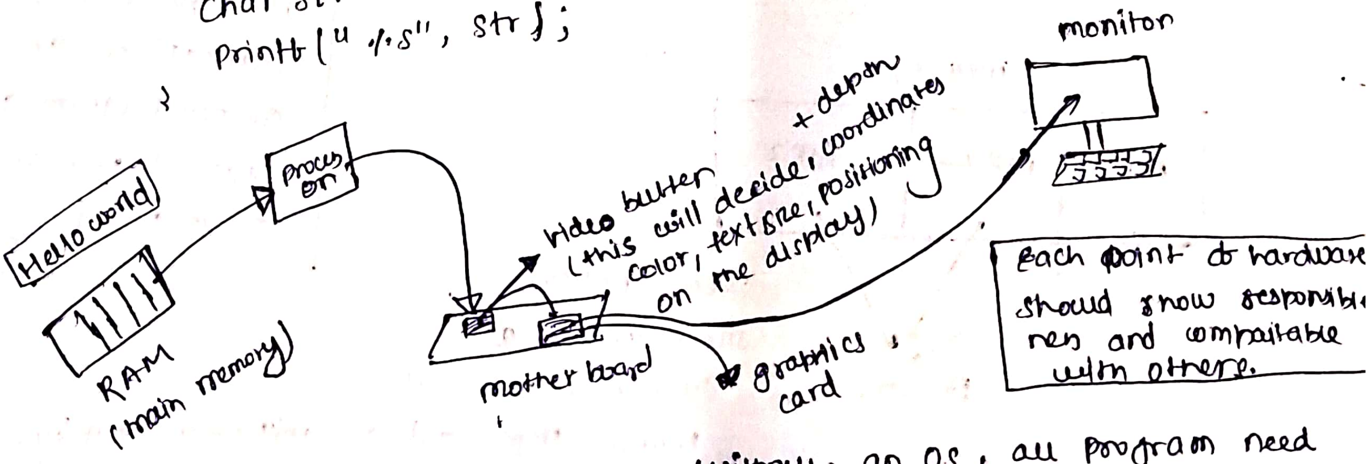
③ Better modularity & better performance

OS usage

① Hardware abstraction
↳ Decide how connection will be happen betwee hardware and ~~s~~ applications without affecting user/ showing.

② mange system resource
↳ As resource is limited. so, utilization and management should be done by a system (OS)

```
#include <stdio.h>
int main()
    {
        Char str[] = "Hello world \n";
        printf("%s", str);
    }
```

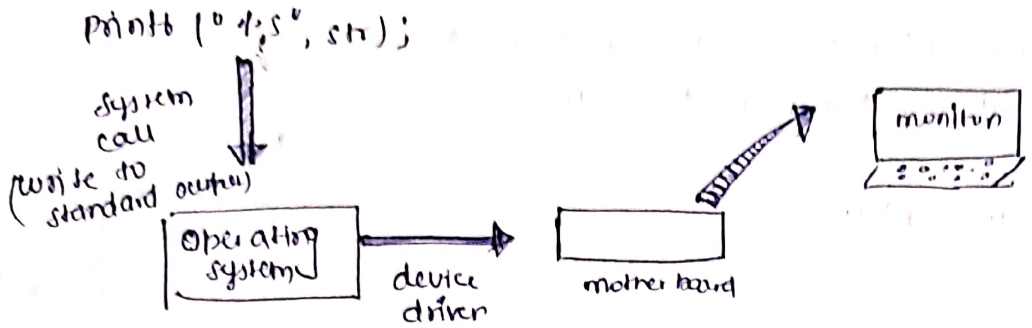

Hello world
RAM (main memory)
Process on
mother board
video buffer (this will decide color, coordinates + depth, text size, positioning on the display)
graphics card
monitor

Each point of hardware should know responsibilities and compaitable with otherse.

- It is complex and tedious.
- Hardware dependent

without an OS, all program need to take care of every nitty gritty detail.

operating systems provide Abstraction



① Easy to program apps → No more nitty gritty details for programer

② Reusable functionality → OS functionality can be reused by apps.

③ portable → OS can even handle when hardware combination get changed. (untill and unless that hardware is design compaitable with OS).

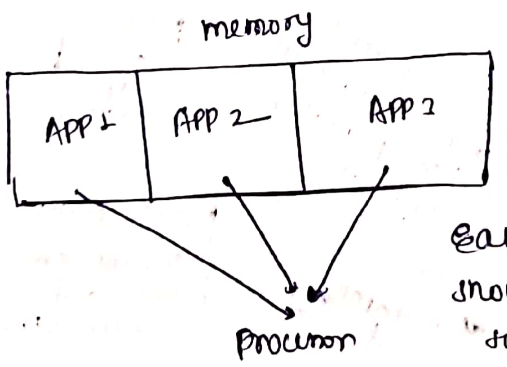
OS as a resource manager

- As hardware is limited but you can have many applications running at a time. So, this ~~increases the~~ is decide by OS that which part will be used when and by which application

- It manage → CPU, Memory, network, secondary storage (hard disk) etc

- Resource management
  - → allows multiple apps to share resource
  - → protects apps from each other interference in resources.
  - → improve performance by efficient utilization of resource

OS helps to prevent starvation. It means if one program get stucked, then it will move to other.



Each application should assigend data such that no interference between them and no milliscious activity.

So each are in sand boxed environment

*Share resource but keep application isolated from each other

Application specific

Battery is main concern.

① embeded OS

- eg. contiki OS, for extremly memory constraints environment

② Mobile OS

- Android, ios, Ubuntu. Touch, windows Touch.

③ RTOS (Real time operating system)

- QNX, VxWorks, RTLinux — used in space mission and other critical mission.

④ Secure environments (used in banking websites)

- SeLinux, SeL4.

⑤ For server

- Redhat, ubuntu, Windows server.

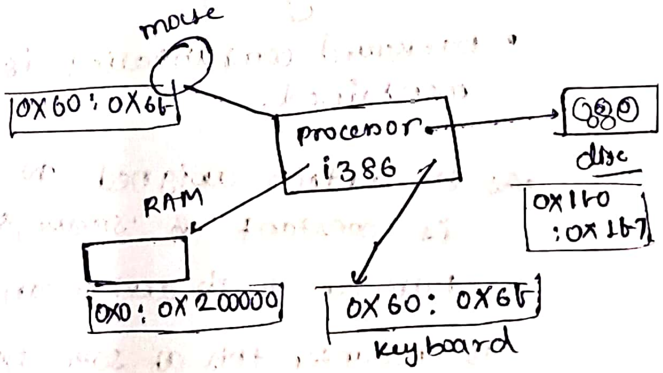
Speed is concern

⑥ Desktops

- Mac OS, windows, ubuntu.

★ XV6 is OS developed by MIT that is easy to understand.

Everything has an address

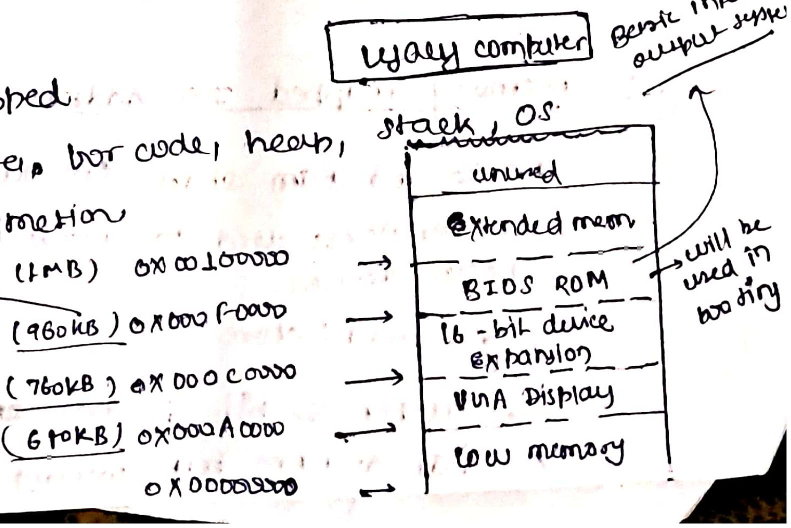


Address type

① Memory address
② I/O address
③ Memory mapped I/O address.

① Memory address

- Range '0 to RAM size
- where main memory is mapped
  ↳ used to store data, for code, heap, stack, OS
- Accessed by load / store instruction



These were used by old computer and not used by current processor

check. KB ✓ or KiB

• If you have 8 GB of RAM then I can use above 1 MB for the application

Address type

1. Memory address
2. IO Address
3. Memory mapped IO Address.

even though lower part of section in RAM has been changed but previous rules are continued still yet. this is legacy aspect.

Memory Address

See, suppose 16 bit is allocated for address, number of address that can be mapped = 2^{16}.

• In general, each value of address maps to 1 Byte storage. we can see RIPES simulator. So, total 2^{16} Bytes can be addressed $= \frac{2^{16}}{1024}$ KB

$= 2^6 = \underline{64\text{ KB}}$

② IO ports (Input output ports)

- Range : $0 - 2^{16}-1$
- used to access device
- uses a different bus compared to RAM memory access. It means it is completely isolated from memory.
- Accessed by In/out instruction
- Backward compatibility is still mentained.

every input-output ports has been assigned same ~~memory~~ address. And O.S will search into those address for that device.

I/O address range	Device
60 - 6F	keyboard
70 - 7F	real time clock
8F	refresh
200 - 20F	game port
F1	reset

• As the address assigned to I/O device is constant & small, so only limited number of I/O devices can be connected.

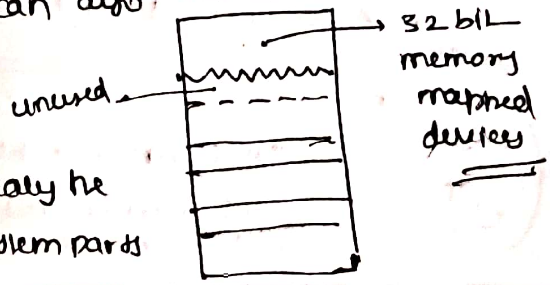
• To encounter this at some extent unused part of RAM started being used to Map I/O devices. and leads to third type of address type

③ Memory mapped IO Address

- Device and Ram share the same address space
- Instructions used to access RAM can also be used to access devices.

→ Eg → Load / store.

So, upper 32 bit is now physically the part of RAM but working for system parts



Who decides the address ranges?

① Standard / legacy

→ such as IBM pc standard and thus backward compatibility is to be followed as chang can cause previous ~~software~~ to Hardware totally useless.

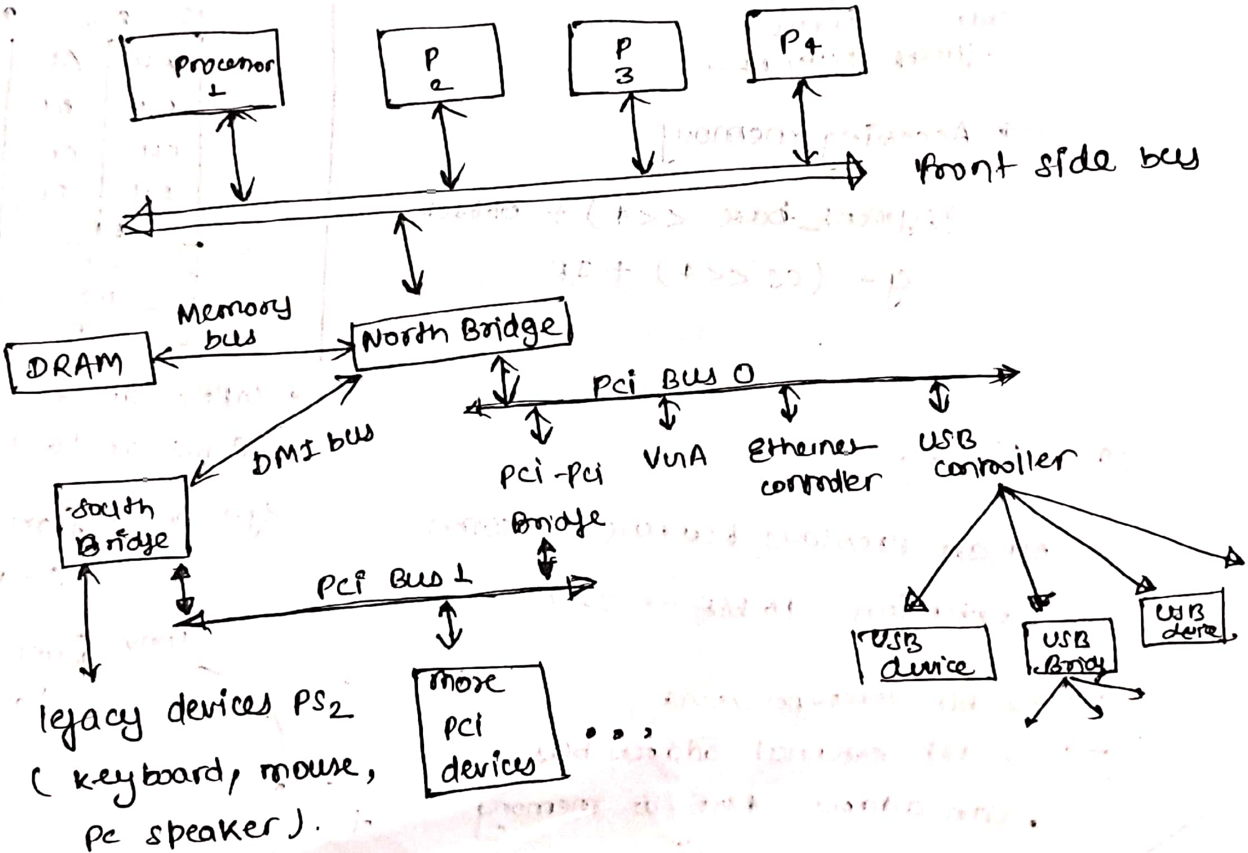
→ Fixed for PCs.

→ This ensure BIOS and OS to be portable across platforms.

② Plug and Play devices

This is decided by BIOS or OS. When BIOS happened in the system then set of addresses get assigned. This restarting address may be different from what was before or previous BIOS of PCs.

PC organization



The X86 Evolution (8088)

- 8088

→ 16 bit microprocessor

→ 20 bit external address bus - So, It can map to = 2^{20} Bytes = $\frac{2^{20}}{1024}$ KB = 1MB

Even though It is 16 bit microprocessor It can addressed to 20 bit because we left shift the address by 4 thus adding 4 mor bit. As 16 bit is reserved or min data type used.. (hw) half word

→ Register are 16 bit

(1) General purpose register

AX, BX, CD, DX

(2) pointer register

BP, SI, DI, SP

Base pointer, starting index, Destination index, stack pointer

(3) Instruction pointer (IP)

segment registers

CS, SS, DS, ES

code segment, stack segment

→ Accessing memory

(segment_base <<4) + offset

eg - (CS <<4) + IP

General purpose register

15 8	7 0	16 bit
AH	AL	AX
BH	BL	BX
CH	CL	CX
DH	DL	DX
BP		
SI		
DI		
SP		

- GPRs can be accessed as 8 bit or 16 bit registers

eg - mov \$0x1, %ah; 8 bit move.

mov \$0x1, %ax; 16 bit move.

(2) 80386

All are previous feature + some extension 16 bit → 32 bit

→ 32 bit microprocessor

→ 32 bit external address bus.

- can address 4GB of memory

→ Registers are 32 bit

General purpose register

extended EAX, EBX, ECD, EDX.

pointer register

EBP, ESI, EDI, ESP.

+ Instruction pointer (I)

+ mov (0x1, %eax); 32 bit move

32 bit

EAX

EBX

ECX

+ more feature

- protected operating mode
- virtual address.

- ~~M~~ AMD (K8) (2023)

→ RAX instead of EAX

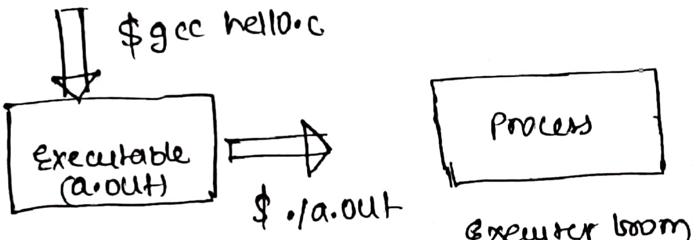
→ X86-64, X64, amd 64, intel 64, all same thing

- Backward compatibility

All system backward compatible with 8088,

Lecture → 3

```
# include <stdio.h>
int main() {
  char str[] = "Hellow world \n";
  printf(" %s", str);
}
```



Executable file will be stored in hard disk only

Executes from RAM. It means peice of code will allocate memory in RAM.

- process

→ A executable program in execution..

→ present in the RAM

→ comprises of
- executable instruction
- stack
- Heap
- state in the OS. (in kernel)

→ state contains : registers , list of open files , related processes etc