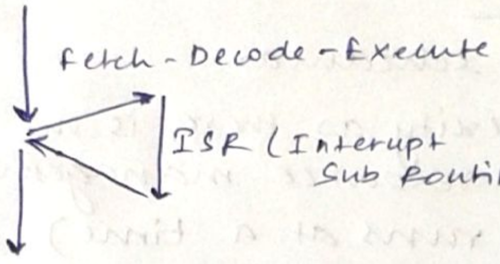
Operating Sytems

* what is it & what does it do?
- manages resources such as memory, processor, i/o, etc.
- provides an abstraction between user & hardware.

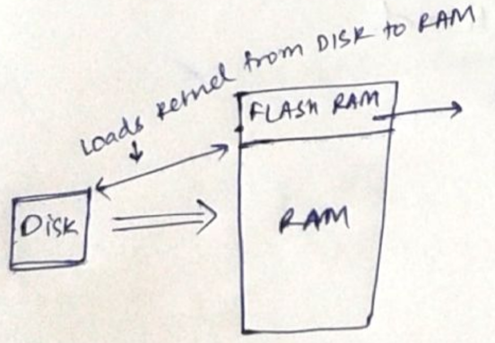
↳ Applications,
(OS)
Hardware



* **Bootstrapping**
* when we start the PC, [PC] gets initialized to some value, but who takes incharge?
  - BIOS

use ROM, problem: can't write new programs
use Flash RAM, problem: Limited Read/write operations

so, use a combination of RAM & Flash RAM?
↳ Advantage over ROM is we can update BIOS.

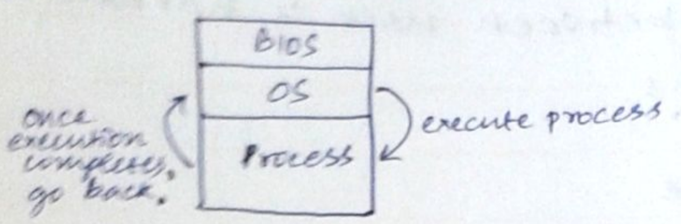


→ First Level Boot Loader, BIOS Routines, ... etc
~~Advantag~~
↳ has access to I/O Routines
(to choose from OS menu)

[PC] will get initialized to memory location in FLASH RAM.
↳ gets initialized to "Reset vector"
↳ contains jump instruction to the BIOS/Boot Loader.

* MS-DOS — Single tasking OS

Memory Map



Advantages

* No need for schedular
* less complexity as there is no need for resource management (1 pgm runs at a time).
* Extract maximum performance from process

MS-DOS games pretty good as per standards

Interrupt vector needed to maintain a timer in case pgm crashes/keeps running, needed to stop pgm.

Disadvantages

* No protection, OS memory can be breached while pgm is running.
* Resources aren't efficiently utilized.

OSV → similar to MS-DOS.

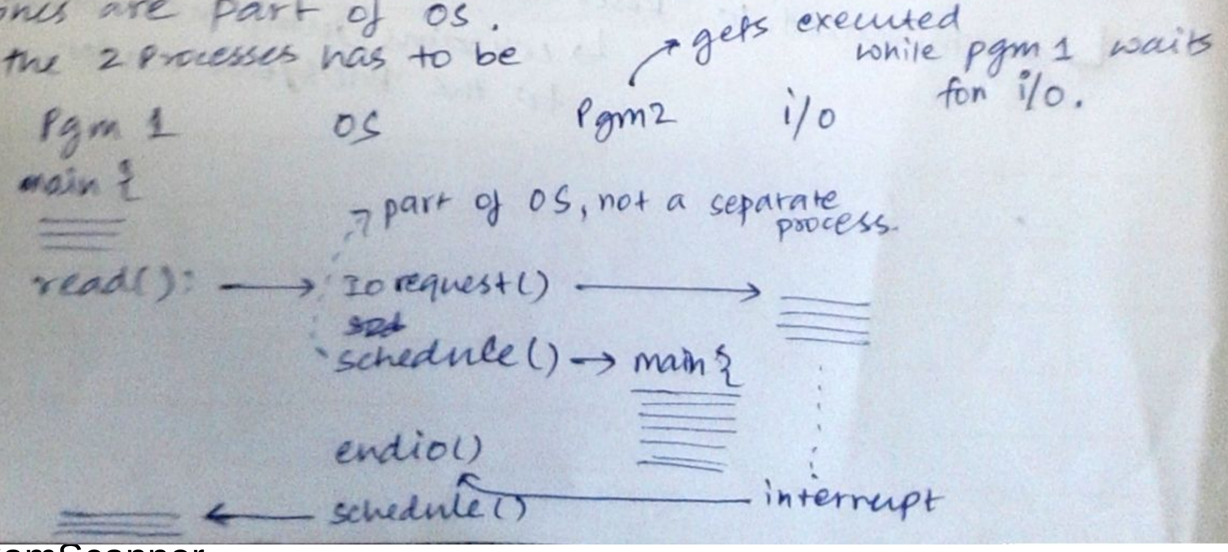
(Multi-Programming?)

* Multi-tasking OS

- Switch b/w pgms based on I/o Requests.

- I/o Routines are part of OS.

states of the 2 processes has to be saved while switching b/w them.



Issues

- kernel isolation
- Pgm isolation
- scheduling
- response time not guaranteed (P1 keeps running without I/o, then no other process will get CPU time)

- Synchronization
  - two jobs executing working on the same file.

* Linux kernel, alone is of no use, we need applications to be able to run with the kernel.
  GNU has made a lot of contributions regarding applications for linux kernel, so also known as GNU Linux

* kernel uses ISA for the processes.
  ISA is an abstraction b/w processor & kernel.

* memory protection for kernel is an hardware implementation

* The kernel once loaded into RAM safeguards itself from user intervention.

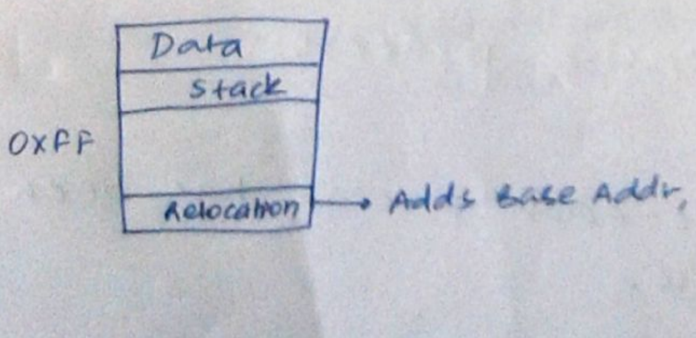
* pgm is static, process is dynamic, i.e, loaded into RAM & running

# Time-Shared OS

- Each process is given a specific CPU time. It is based on interrupts.

Two Pgms can access same memory location. Problem?

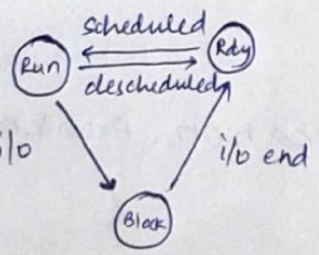
Two processes P1 & P2
each has a memory segment



When a pgm is allocated memory, the compiler assumes a address 0x0 & assigns the addresses to every variable (by incrementing from 0x0). However, this is not the actual address, so the loader knows the Base address of the pgm & then for every variable add Base Address & address given to it by compiler.

compile time	load time	Run time
a.out (no base addr at compile time).	[Base Addr] + Relocation table [indexing for assumed addr].	

* <u>Process scheduling stages</u>



Scheduler chooses b/w pgms in the ready state.

switching process b/w machines requires ~~a~~ complete information of the stopped process such as memory map, processor register values, stack, heap, etc.

* <u>Time sharing OS</u>

Pgms can be switched once each process has been executed for a specific time.

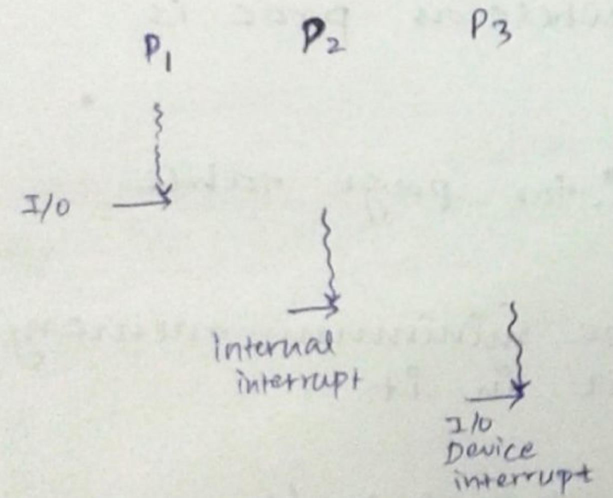
Kernel → combination of data & code.
Scheduler uses info. stored by kernel to make decisions.

* Virtual Memory

Each pgm is given an illusion that it has the whole CPU & memory to itself.

Two processes running, having same "data" variable in "data segment" (global segment), is given same address (virtual address), however actual address is different ~~a~~ mapped for each process by something called "page" tables.

However, local variables are given different addresses in stack due to buffer overflow protection.

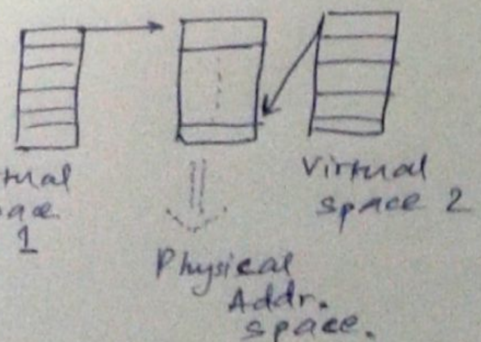


No performance Isolation
↳ No matter how many processes run, the processor can execute a limited no. of instructions at a time.

Every process have access to complete virtual address space however not full physical address space.

* pgm isolation.
* kernel isolation.

* saves overwriting of memory of P2 by P1.



Two processes can be mapped to same physical addr space if those two processes share the same ROM segment. such as data / code.

Every kernel has a Data structure.

For eg:

u-area (accessible when process runs)

proc. → used by schedules to know about process. (accessible ~~and~~ time).

Every system call uses a part of what is called "kernel stack", whereas process running uses ~~[illegible]~~ "user stack".

~~[illegible]~~

u-area is present in user space where as proc is present in kernel space.

* Easy for kernel to access "proc". for page table information.

kernel space is meant to utilize minimum memory so no point of putting u-area in it.

Two modes exist for the processor, kernel Mode, user Mode.

fork()

↳ creates a copy of current process.

↳ copying is an expensive process

*copy on write

$$\begin{tikzcd} & \text{mechanism (How does it schedule?)} \\ \text{Process scheduling} \arrow[ur] \arrow[dr] & \\ & \text{Policy (when should you call scheduler?) (which process?)} \end{tikzcd}$$

Scheduling policy: optimize some metric

Batch apps. - Turn around time [t_c - t_a] → completion time.

Interactive apps - Response time [t_f - t_a] ← t arrival time

allocated CPU

online scheduling means the data comes in as the algorithm / scheduling starts to run.

FIFO, whoever comes first execute that, however a problem exists if a large ~~problem~~ process comes first.

starvation problem with STCF

Process priority = base priority + nice · value + (* recent CPU usage * / constant)
↓
Forgetting factor
↓
used to remove history