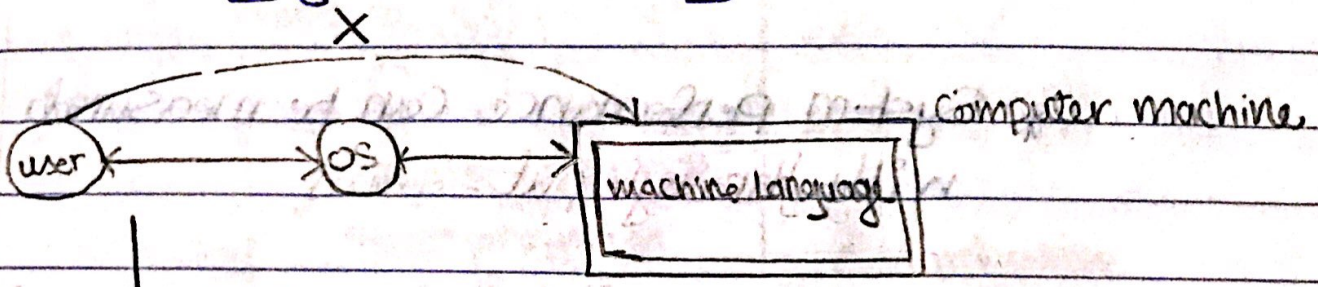
Chapter#1
Introduction

lecture#1
Feb.10.2018

Operating Systems Goals:

(1) overall goal: Execute user programs.
(2) primary goal: Conveniency



↳ it's easier for the user to interact with the OS.
(3) Secondary goal: efficiency.

Computer Resources:

(1) CPU.
(2) Memory.
(3) I/O devices.

} OS manages Resources.

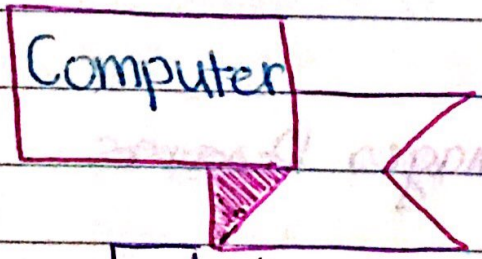
Operating System (OS): A set of Algorithms that run the Computer machine.
↳ OS manages the Computer resources.
↳ OS must manage the Computer Resources efficiently.

→ Other goals of OS:

- Utilization of Computer resources.
  - Utilization of CPU: make the CPU as busy as possible.
  - Utilization of Memory:- to benefit or use memory as much as possible.
  - Utilization of I/O devices.

* System performance can be measured with throughput.

* Throughput: number of jobs (programs) that finish execution per unit of time.



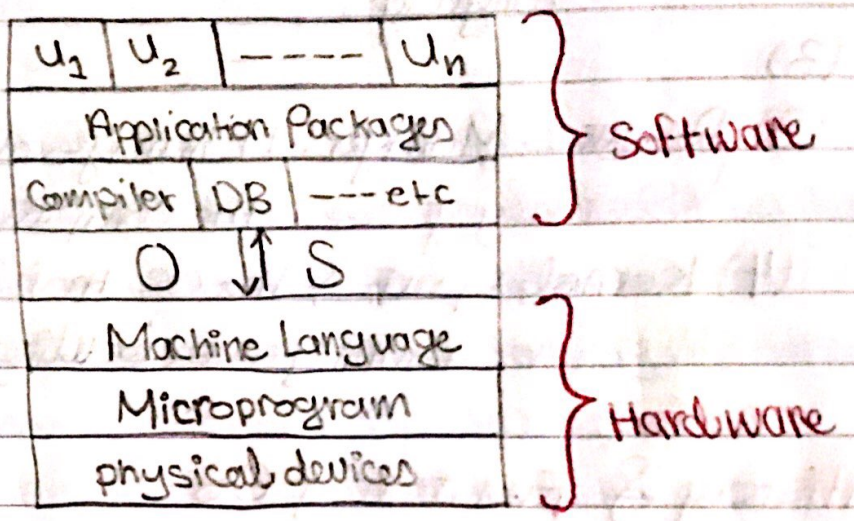
1. HARDWARE:-
   - physical devices: chips, wires, Power supplies...
   - micro program: A primitive software that communicates with Physical devices which is an Interpreter that fetches

language

→ Fetch: execute machine instruction.

— machine Language (Assembly Language)

2 OPERATING SYSTEM:



3 APPLICATION PACKAGES:

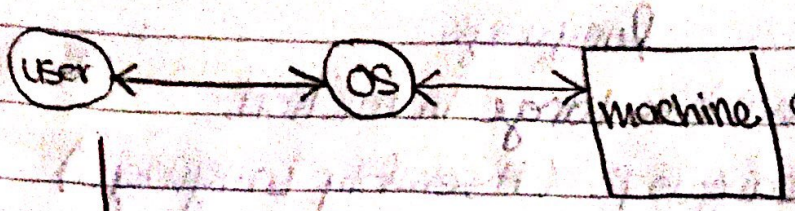
4 User Programs.

Operating System Views: OS Goals

(1) It's a control Program: It controls the execution of all programs.
↳ overall goal.

(2) Extended machine: Extension of the physical machine.
↳ primary goal.

That is it hides all the complexity of system programming and provide the user with a simple clean machine to deal with

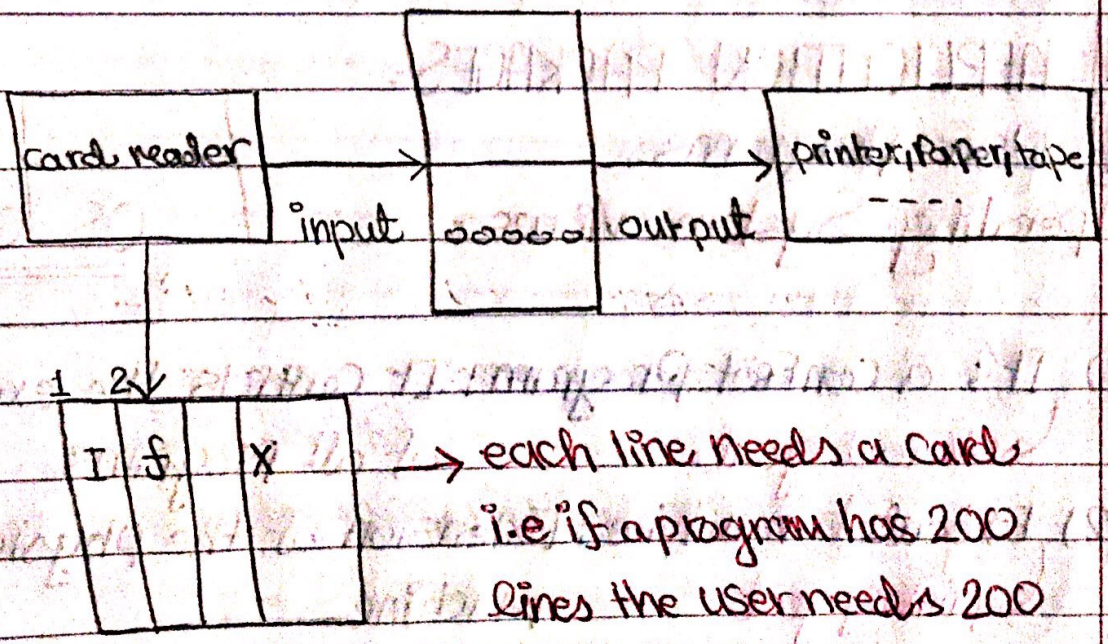


↳ The user doesn't have to deal with machine or use assembly / machine Language..

(3) Resource Manager: It manages efficiently the computer resources.
↳ Secondary goal.

(4) Kernel: part of the OS that's always running and executing instructions

History & Evolution of OS :-



→ each line needs a card
i.e if a program has 200 lines the user needs 200 cards.

* Hexadecimal was used.

[+] Early software :-

- machine Language.
- Assembly Language. (Assemblers)
- Loaders
- Linkers: the addition of software to program.
- Compilers.

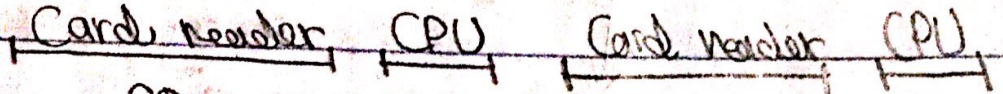
* Performance is poor
  - a great deal of time is wasted in set up time.
  - No overlap between I/O & CPU execution
  - Low CPU utilization (due the big difference between I/O speed & CPU speed)

example:

A fast card reader can read 1200 cards/min

1200/60 = 20 card/sec.

The CPU can process 300 cards/sec.

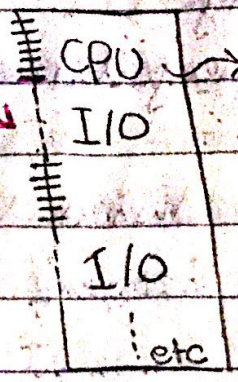


∴ Percentage of CPU utilization

= 4/64 ≃ 1/16 ≃ 6%

Lecture #2

Feb. 12. 2018
Monday

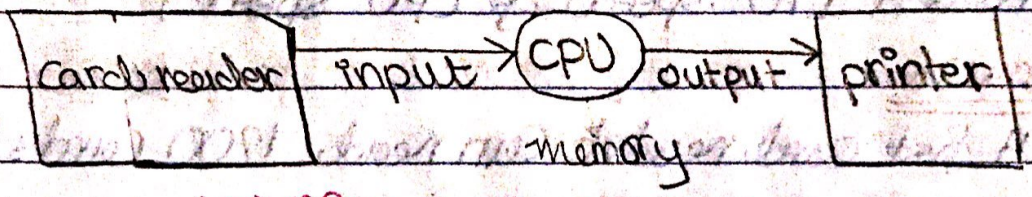


CPU Instruction

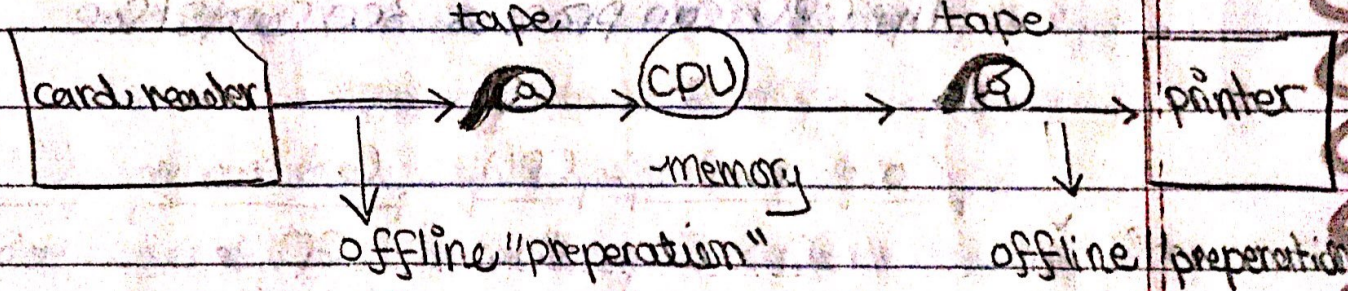
⚠ CPU instruction executes until reaching I/O
↳ printing or reading input.

[A] offline Operation:

(1) Before

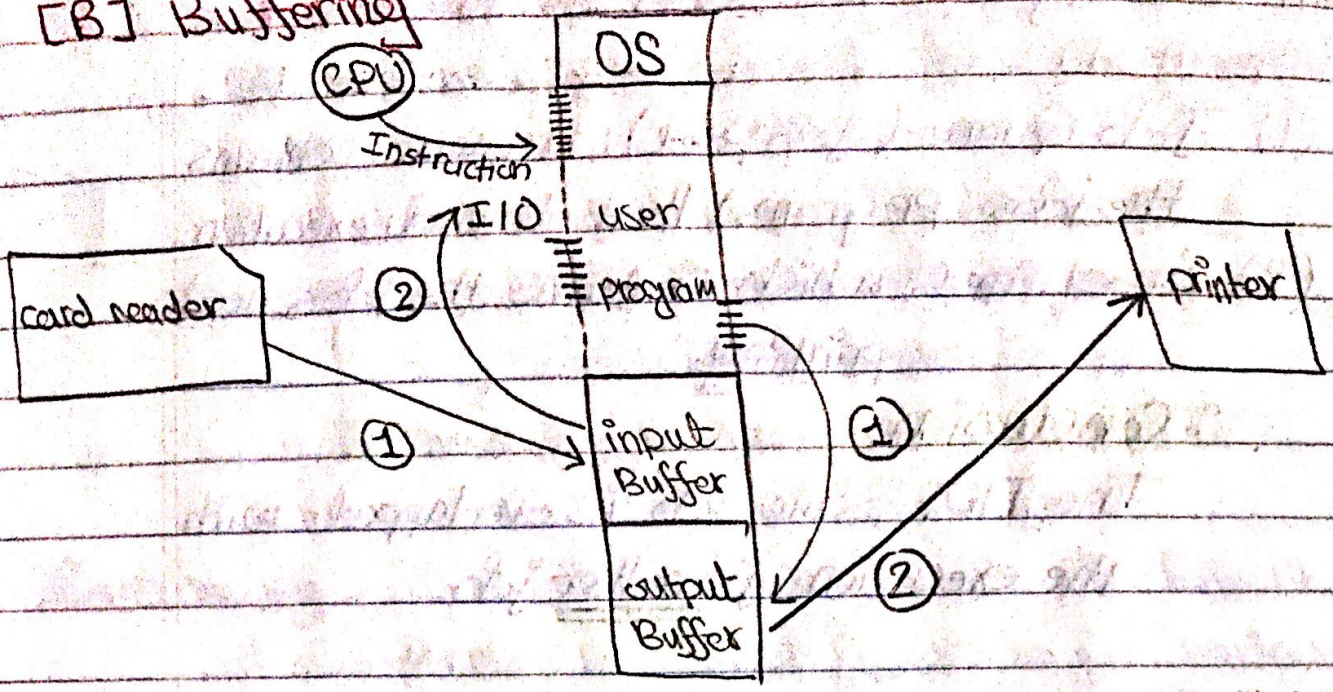


(2) After



⚠ Tape to memory is much more faster than Card reader to memory.
↳ improve execution.

[B] Buffering

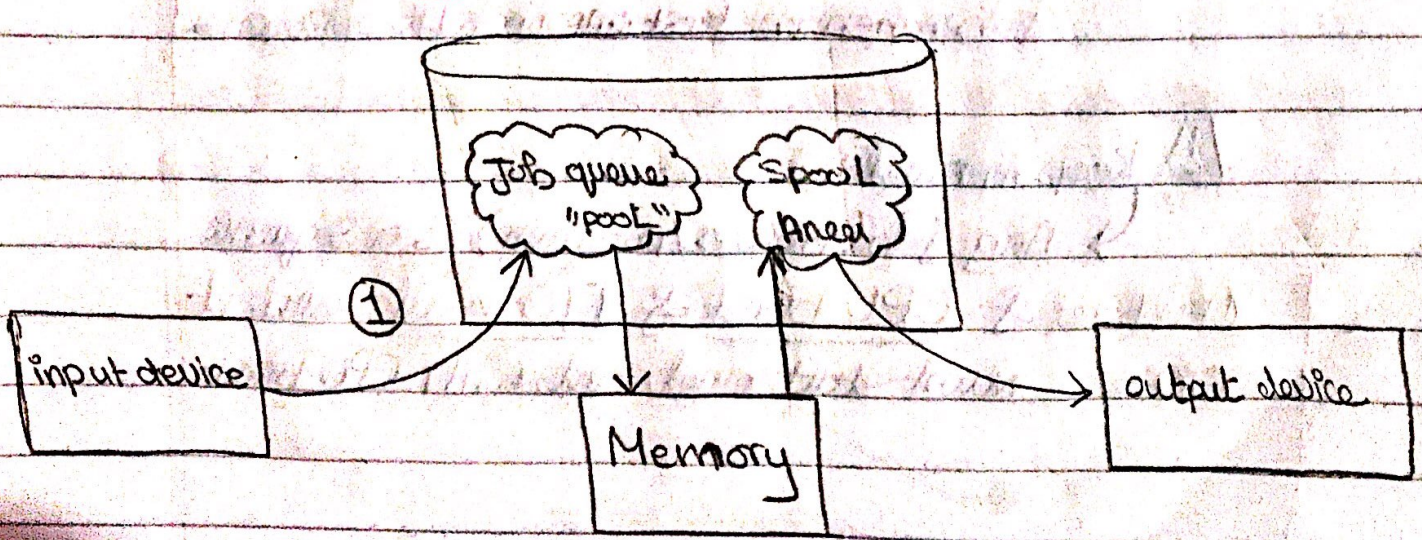


* when CPU reaches I/O it brings data from buffer not from the card reader

∴ Conclusion:

the I/O of one job is overlapped with the execution of same jobs.

[C] Spooling



In Spooling 2 kinds of data structures were introduced:

(1) Job Queue (Job Pool): A queue contains the jobs (programs) that demands execution.

(2) Spool Areas which contains the jobs need printing.

∴ Conclusion:

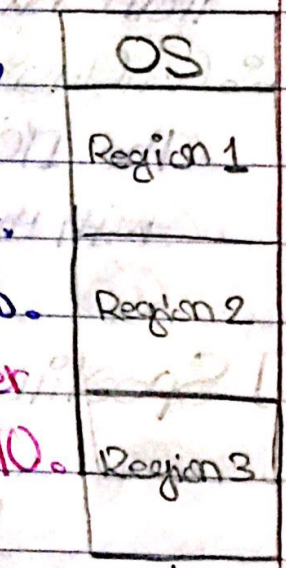
The I/O of one job is overlapped with the execution of another job.

[D] Multiprogramming Batch Systems

"Multi-programming"

- Memory is divided into several Regions (Partitions).
- Regions sizes normally different.
- Each region contains only one job.

* The CPU switches to another job when the first one needs I/O.



⚠ Keep in mind:

↳ Any job (process/program) is a sequence of CPU burst & I/O wait and it must start and end with CPU burst.

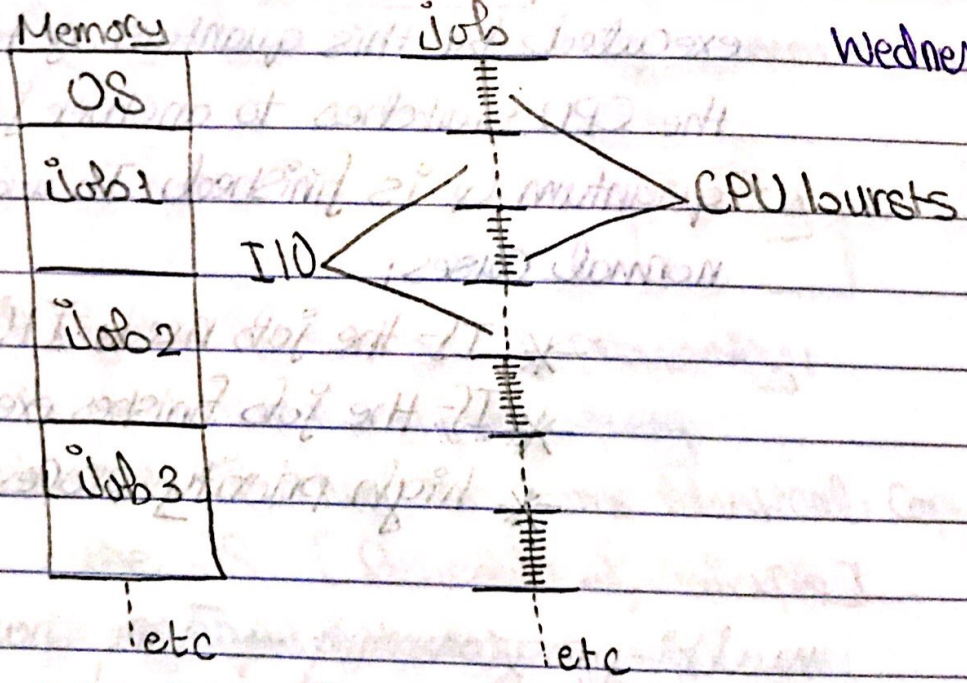
Two kinds of jobs:

(1) CPU-bound job: Contains few very long CPU bursts.
→ most of the time, job needs CPU.

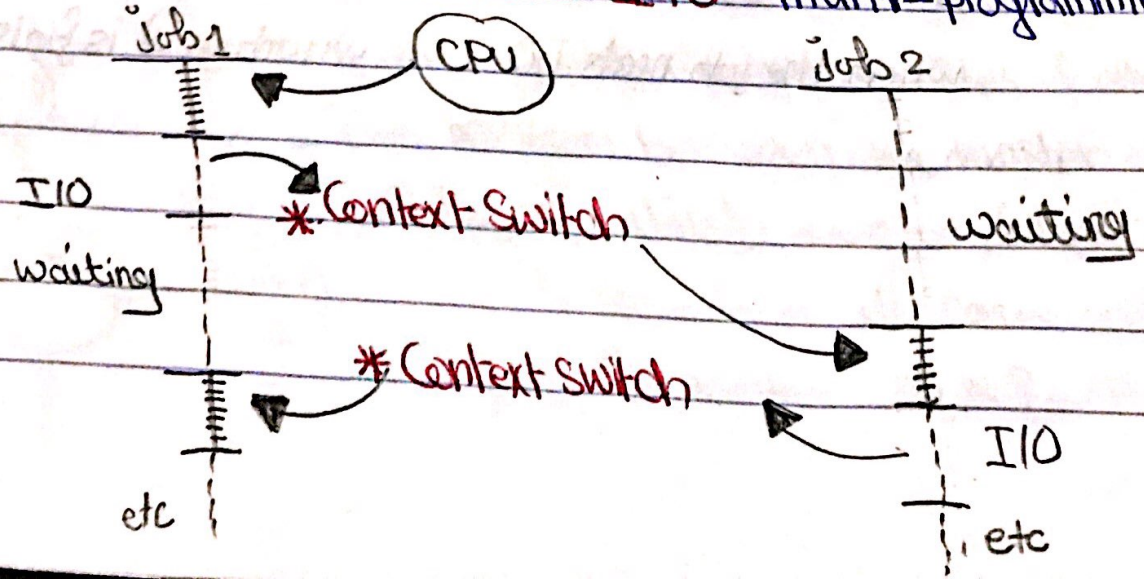
(2) I/O-bound job: Contains many very short CPU bursts.
→ most of the time, the job needs I/O.

Lecture #3

Feb. 14. 2018
Wednesday



⚠ CPU switches (changes) to another job when the first one needs I/O "multi-programming"



* Context switch: Saves Register for job 1
Reloads Register for job 2.

[E] Time Sharing Systems:

- Same as multi-programming.
- Memory is divided into regions.
- Several jobs are kept in memory.
- Each job is assigned a slice of time called quantum Q. Each job is executed for this quantum of time & the CPU switches to another job when quantum Q is finished. In addition to normal cases:
  * If the job needs I/O.
  * If the job finishes execution.
  * high priority process.

multi-programming ≠ Time Sharing

↓ The CPU switches when the job needs I/O

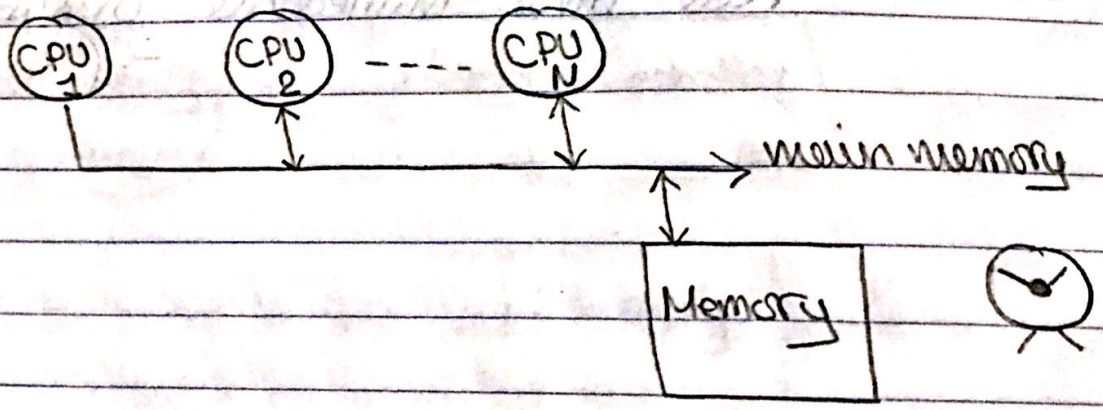
↓ The CPU switches when Quantum Q is finished.

[F] Parallel Systems:-

Multi-processor systems with more than one CPU in close communication.

(1) Tightly Coupled Systems:

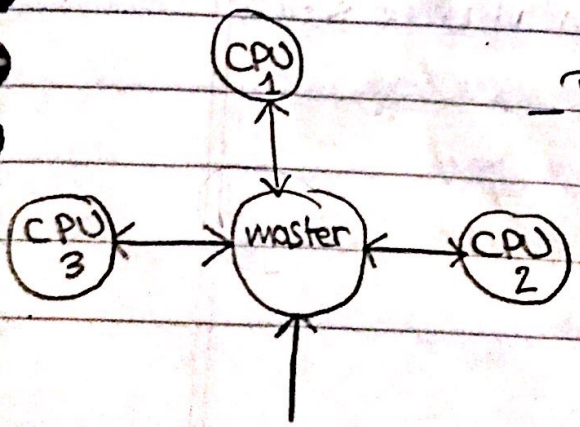
processors share memory, clock, & communication usually takes place in memory



* There are two types of processing:

a- Symmetric multi-processing
↳ Each CPU has the same identical copy of the OS [Reliable & Simple]

b- Asymmetric multi-processing
- There's a huge OS that runs this scheme.
- There's one CPU called master CPU which controls other activities of other CPUs
- The relation between the master and other CPUs is called master/slave relationship
  ↳ Reliable on all cases unless the master CPU is faulty.



(2) Loosly Coupled Systems:

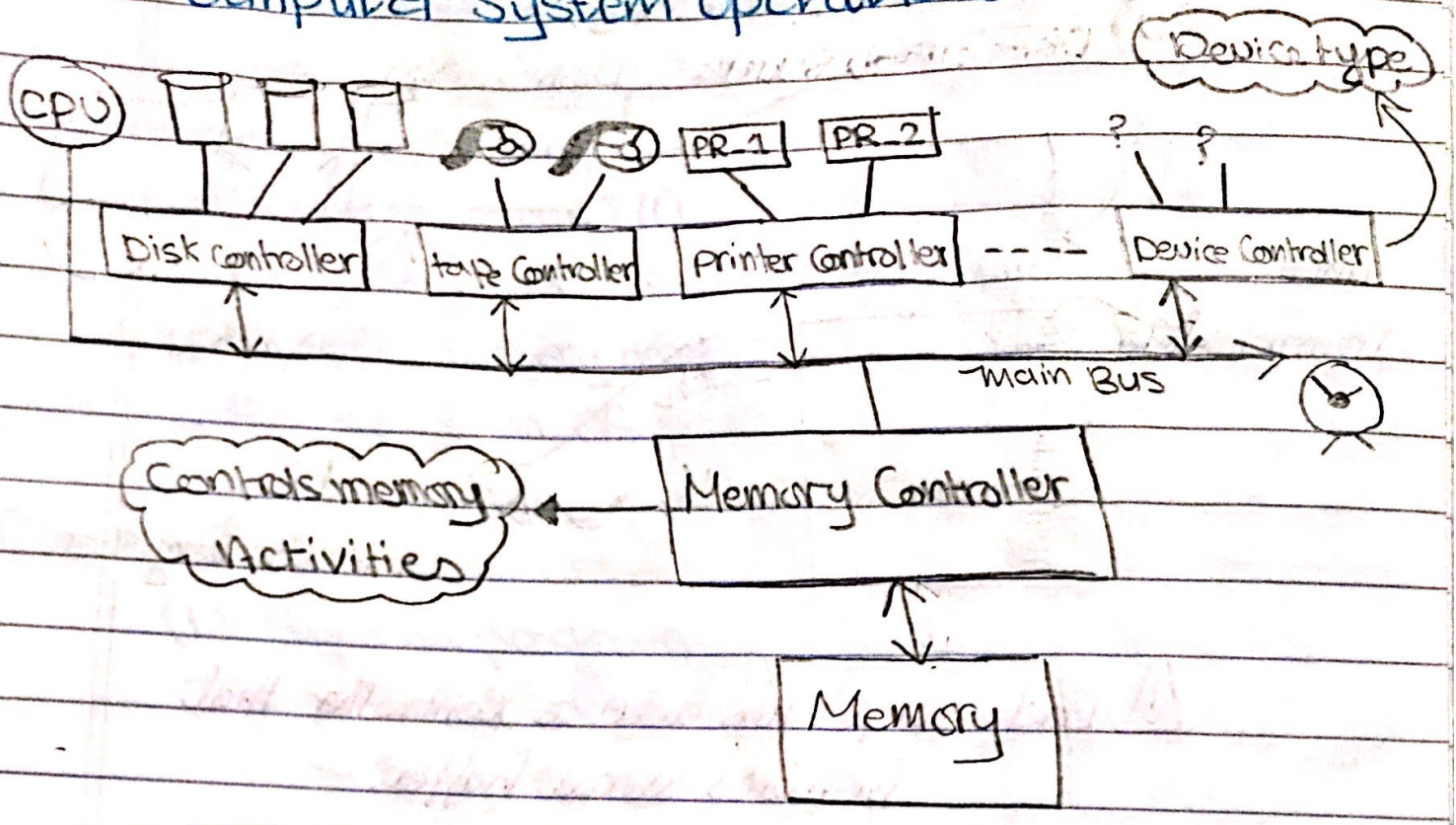
- Networks
  - "Distributed Systems"
- كل نظام قائم بذاته
  - Connect via servers.

[G] Real time Systems:

- takes data using Sensors.
- for systems that need Response real time "immediate" استجابة مباشرة

Chapter#2

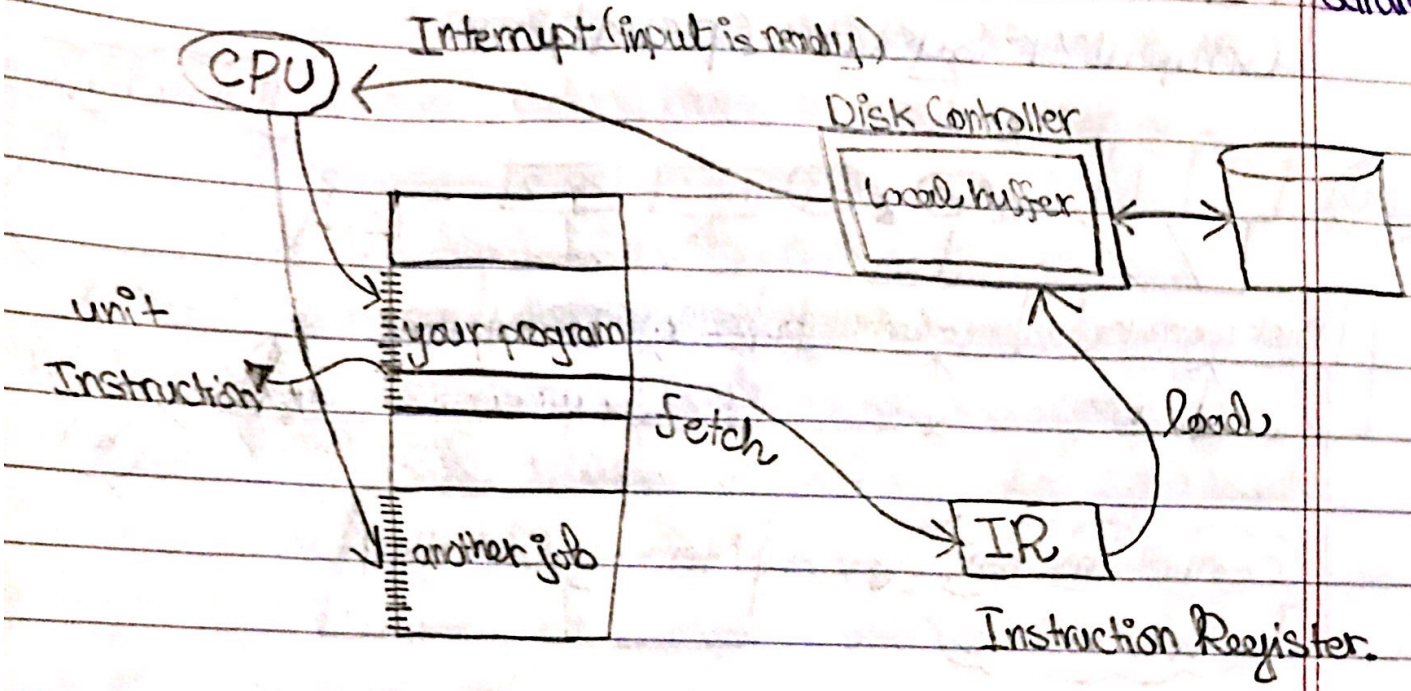
Computer System operations.



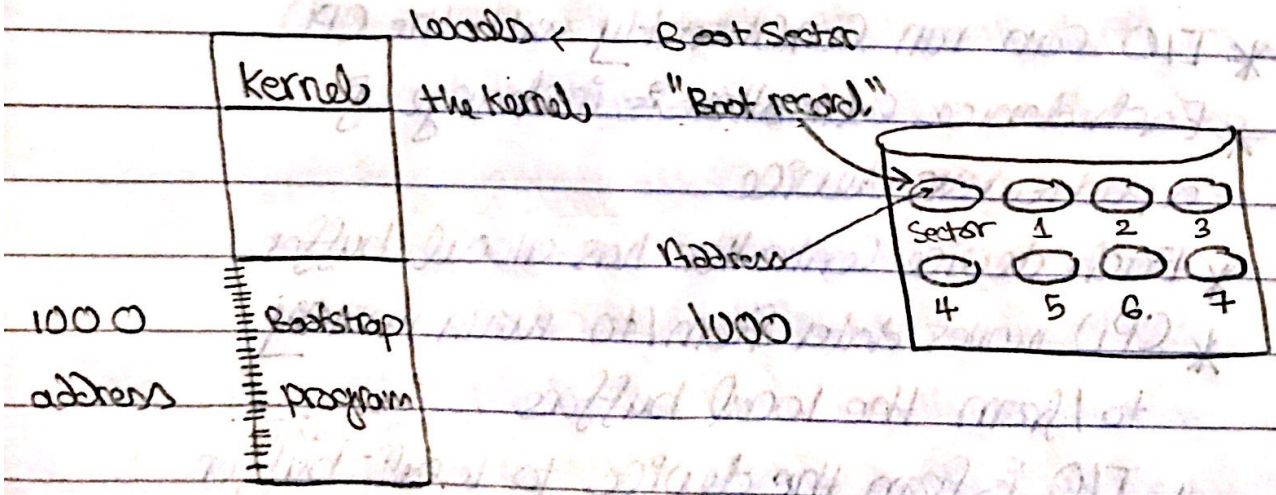
* I/O can run concurrently with the CPU.
* Each device controller is incharge of a particulars device.
* Each device controller has a local buffer.
* CPU moves data from/to main memory to/from the local buffers.
* I/O is from the device to local buffer of controller.
* Device controller informs CPU that it has finished its operation by causing an interrupt.

Lecture #4

Feb. 17. 2018
Saturday



⚠ Each device has a device Controller that includes a local buffer.



⚠ Operating Systems are Interrupt driven (CPU is interrupted)

Interrupt: A Signal sent to the CPU by: "System Calls"

- Hardware
- Software "Trap"

examples:

- Completion of an I/O. "Hardware Interrupt"
- Division by zero. "Software Interrupt"
- Invalid memory access. "hardware Interrupt".
- Request of an OS service.

OS services can be asked:

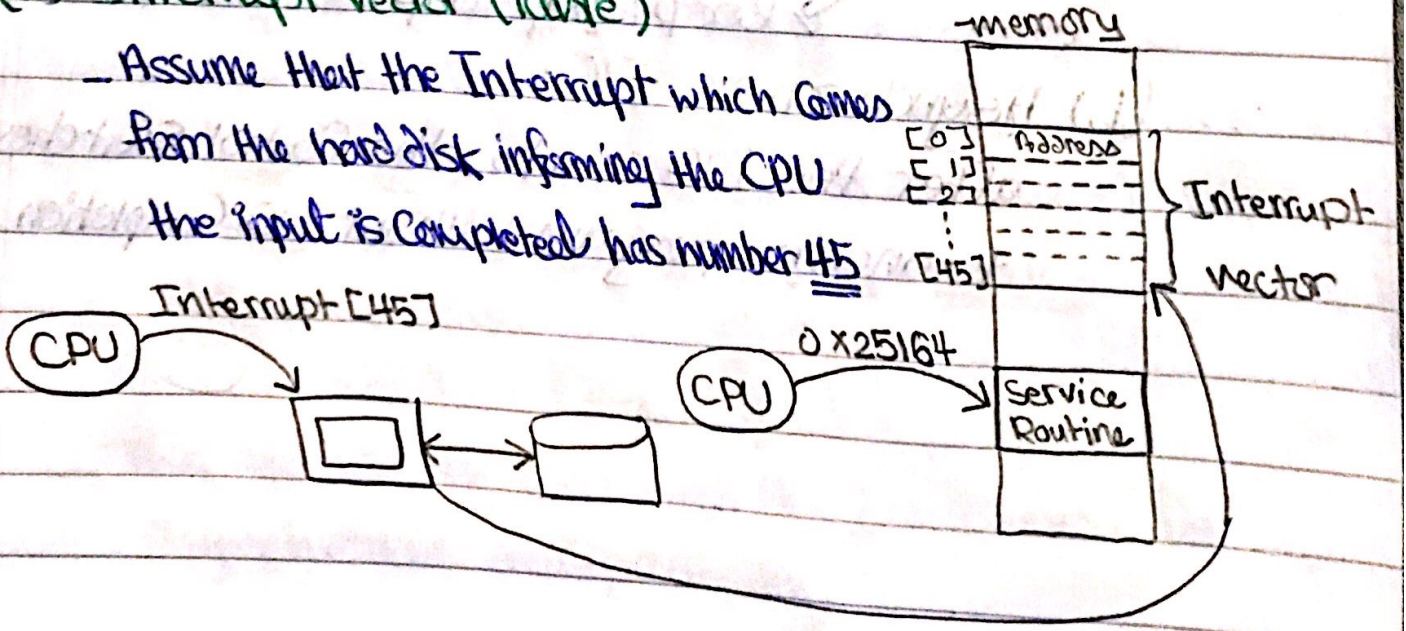
(1) System program
> Format a:
> Copy A.dat B.dat.

(2) System Calls
It's an assembly language Instruction.

Interrupt Handling:

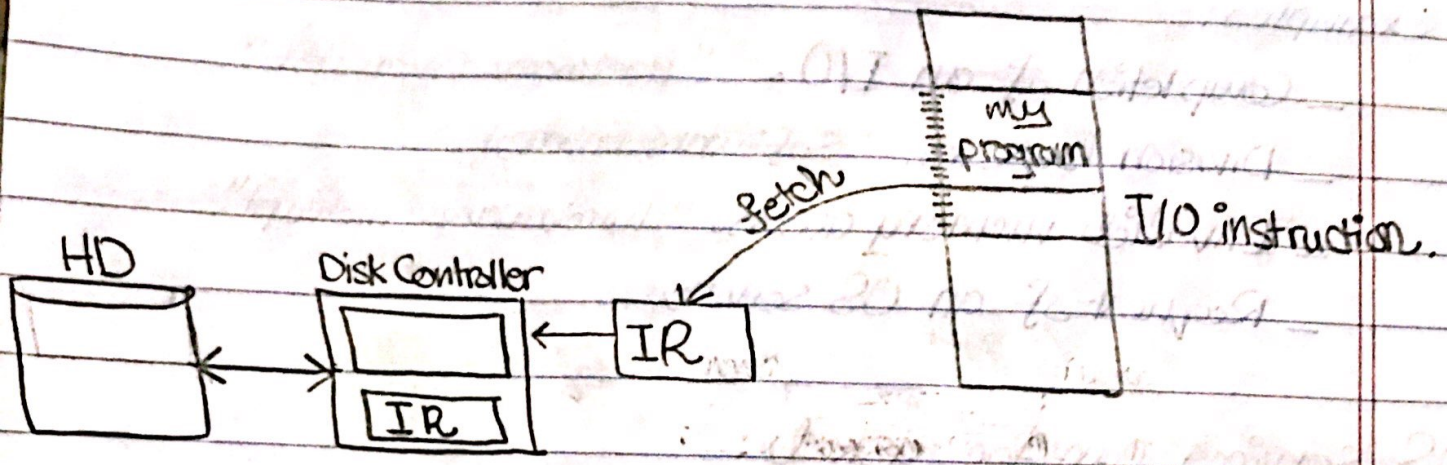
(1) Interrupt vector (Table)

- Assume that the Interrupt which comes from the hard disk informing the CPU the input is completed has number 45



(2) By polling:

I/O interrupt structure:



There are two types of I/O:

(a) Synchronous I/O:

after the I/O starts, the control returns to program only upon I/O completion.

→ The CPU waits until the I/O is completed.

↳ wait Instruction "CPU is idle"

↳ loop: jmp loop

(b) Asynchronous I/O:

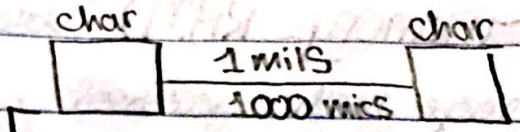
after the I/O starts, the control switches to another program without I/O completion.

Direct Memory Access (DMA):

- Slow I/O devices such as keyboard, can send one character every one millisecond. 1 mils = 1000 mics.
- The CPU can process the character in 2 micro seconds through an I

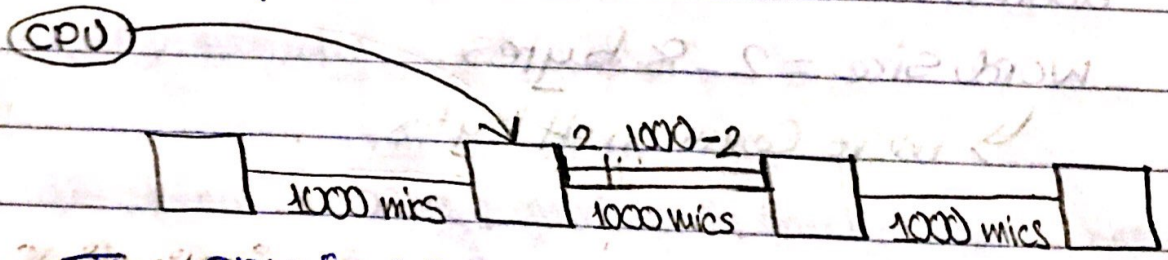
Lecture #5

Feb. 19. 2018
Monday.



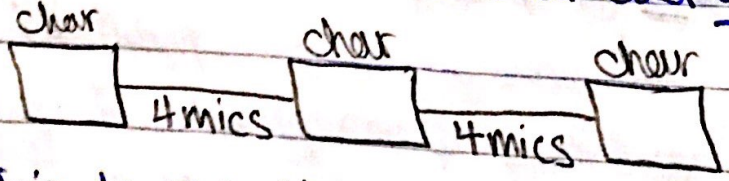
↳ Sends 1 char every 1000 mics.

→ The CPU needs 2 mics for the service routine to handle the interruption.



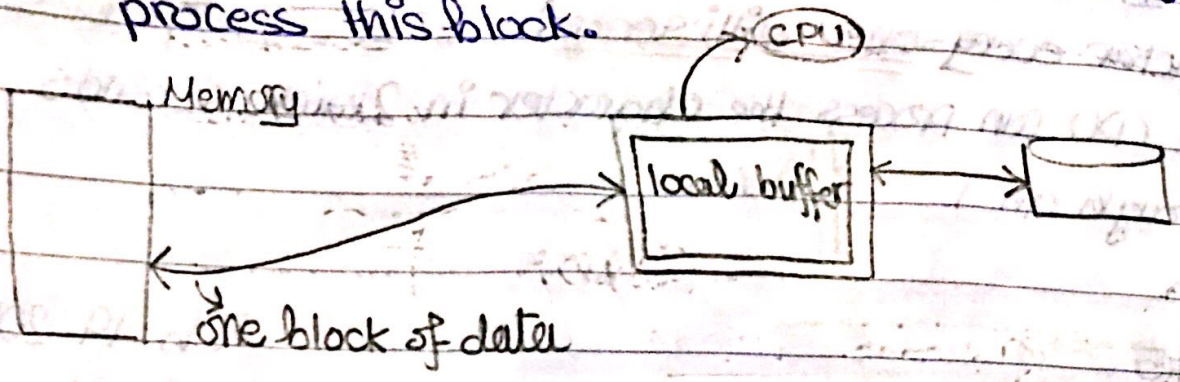
- The CPU is left 1000 - 2 = 998 mics for Asynchronous I/O processing.

- But, In high speed I/O devices (i.e Hard Disk) The HD can send or recieve char every 4 mics.



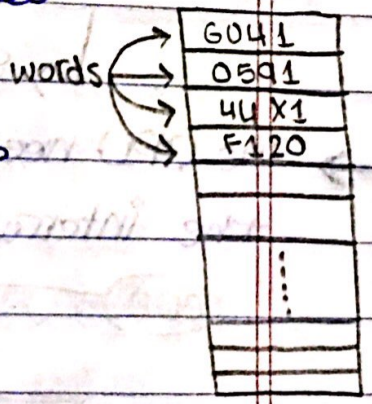
- This leaves the CPU with 4 - 2 = 2 mics for Asynchronous multiprocessing.

* In this case the OS sends one block of data each time & sends an interrupt to the CPU to process this block.



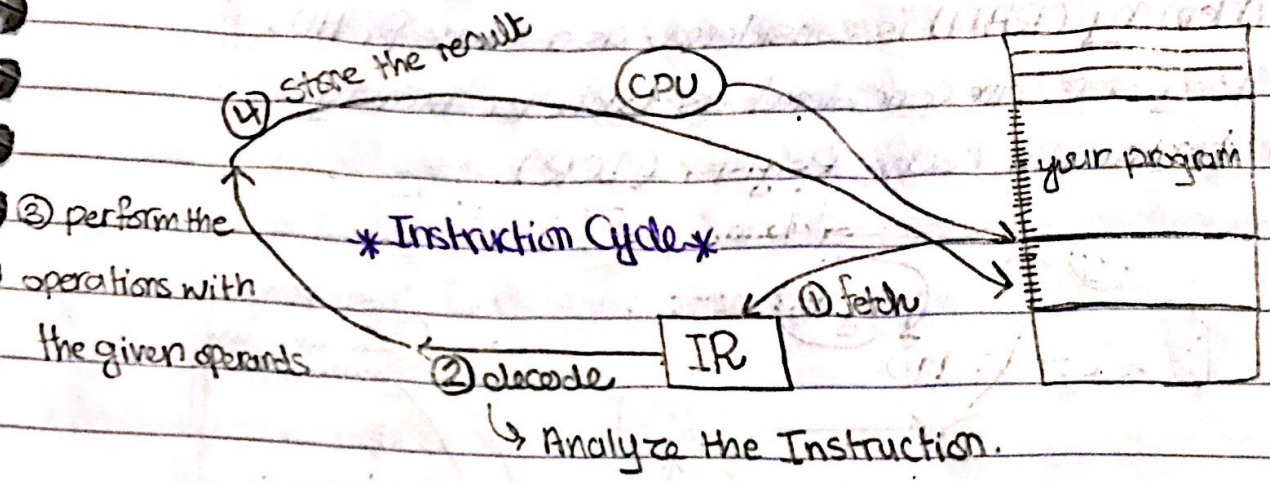
Primary Storage (memory_RAM) "volatile"

- Memory is largest storage media accessed directly by the CPU.
- Memory is array of words, each has address.
- word size = 2 & 8 bytes
  - ↳ more common 4 bytes



- The CPU performs the following instructions:-
  - (a) Load instruction: Fetch (get) instruction from memory to the IR (Instruction Register)
  - (b) Store Instruction: storing a Register in to memory location.

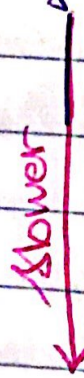
Instruction Cycle:



Secondary Storage: "Hard Disks, tapes, CDs, Flash memories ..."

- Factors that affect the secondary storage:
  (1) speed.
  (2) Cost.
  (3) Volatility. "موقت/دائم"

- The fastest storage medium in the system:



- Register
- Caches
- RAM
- HD
- Tapes
- etc.

* Cache Memory:

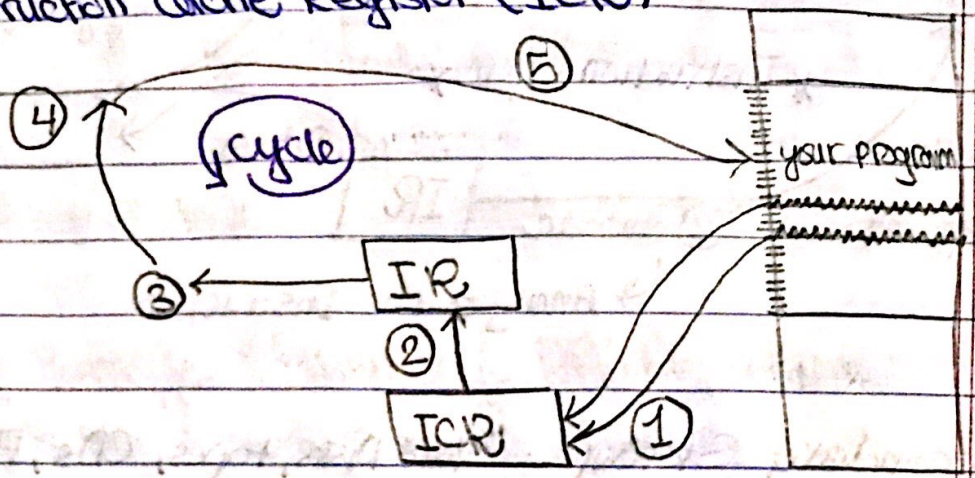
Chashing: is simply copying data to a faster storage medium to speed up execution & ensure good performance.

Examples:

(1) Memory (RAM) is considered as a Cache for HD.

(2) Registers are considered as Cache for memory.

(3) Instruction Cache Register (ICR)



⚠ ① & ② are Concurrent [Parallel].

- from ICR to IR is much faster than RAM to IR

- L_1 Cache is the fastest Cache type because it's build within CPU.
  - L_2 bigger but slower.
    - etc.

Lecture #6

Feb. 21. 2018
Wednesday

[1] Hardware Protection:

How the OS protects the Computer Resources?

- I/O devices.
- Memory.
- CPU.

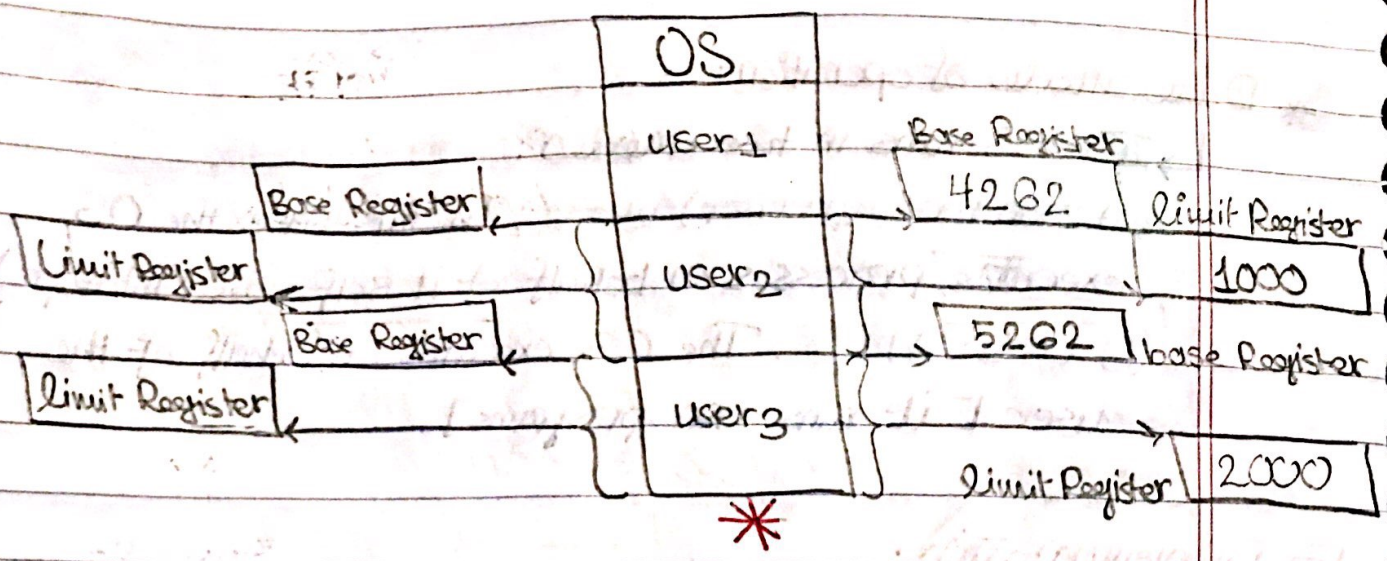
* Dual mode of operation:
  ↳ The OS runs in two modes:
  (a) Monitor (Supervisor) mode: In this case the OS executes process on behalf of it self. (i.e interrupts)
  (b) User mode: The OS executes on behalf of the user [it runs user programs].

Implementation:
↳ one bit is assigned called "mode bit"
0: monitor mode.
1: user mode.

Definition:
A Privileged Instruction is an instruction that's executed only in monitor mode. "OS"

I/O protection:
All I/O instructions are made Privileged Instructions.

Memory Protection:
To protect the memory Allocation space of user program & the OS it self.

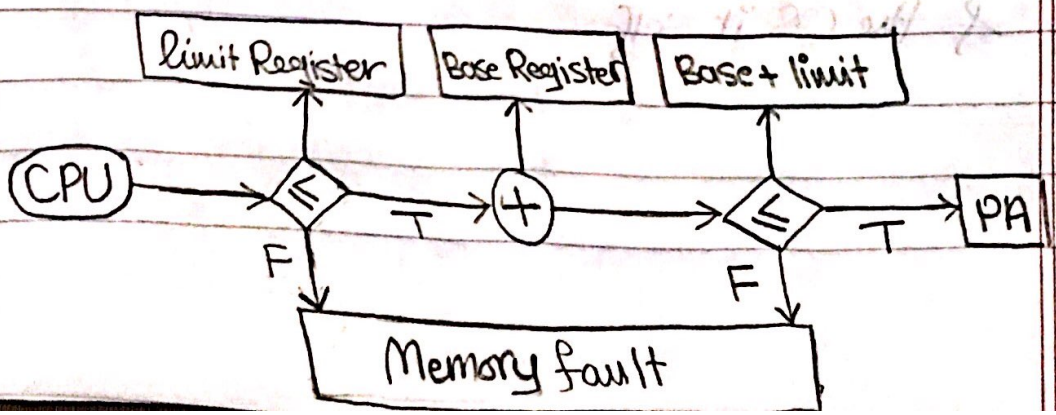


(LA) ← User 2 program → Loaded to the memory * where 738 address starts from 4262 & has a limit of 1000

* Logical Address (LA): is the offset of the instruction in your program.
& LA = the address seen in your program.

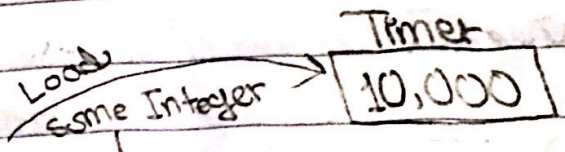
* Physical Address: is the actual address in memory
& PA = Logical Address + Base Register.
(i.e) PA = 738 + 4262 = 4998

➡ The most Important Concept; is How the OS Compute the PA?

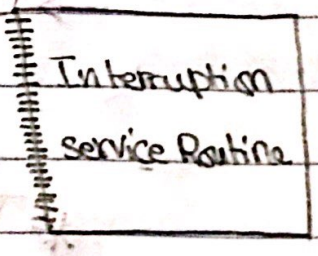


CPU protection:

- Timer



- With every clock tick the timer is decremented by 1
- When the timer reaches zero it sends an interrupt to the CPU.
- When the CPU recieves interruption it executes the interruption service routine which is responsible for checking the CPU.



- Timer can be used in Computer time Calculation.

Operating System Structure:

(1) process management.
(2) Memory management.
(3) File System (I/O) management.

→ The OS services can be provided in two methods:
- System Call: it's an assembly language Instruction.
- System programs.

Lecture #7

Feb. 24. 20[illegible]
Saturday

Process

* Process Concept:-

→ Definition: A process is a program in execution

Process ≡ program ≡ job

* There are different process types in the System:

(1) Batch Process:

generally, Batch process has low priority.

(2) Time Sharing process:

- users.
- program development.
- data entry.
- gamming.

(3) System tasks:

- Interrupt

* Process Contents:

The process contains:

(1) Code Section (Program Counter "PC")

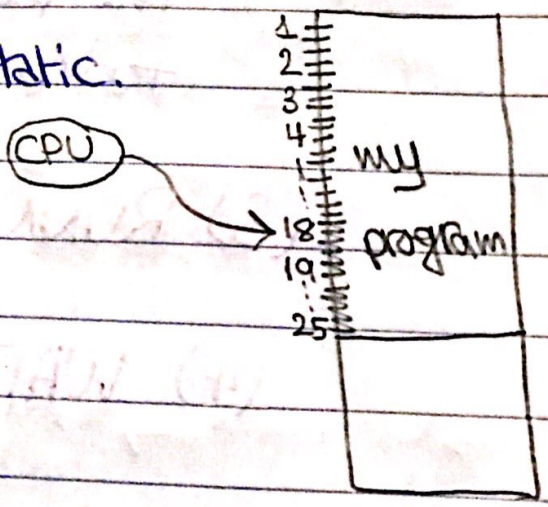
"PC": is the address of the Instruction executed currently.

(2) Data Section:

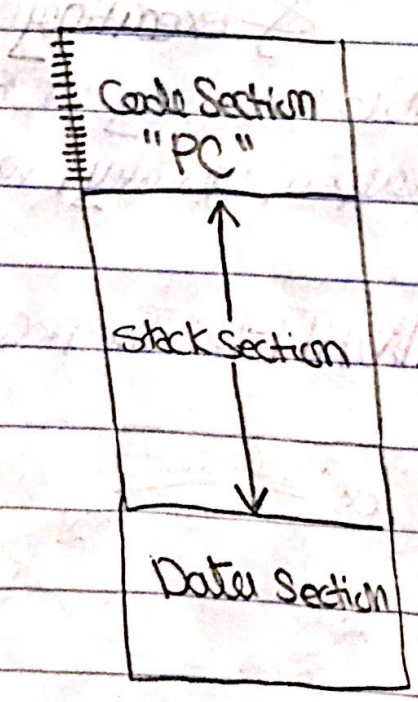
Memory allocation were Input & Output data is stored.

(3) Stack Section:

- Global variables: Static.



(a)



Process.

* Process States:

(1) NEW: The process is newly created.

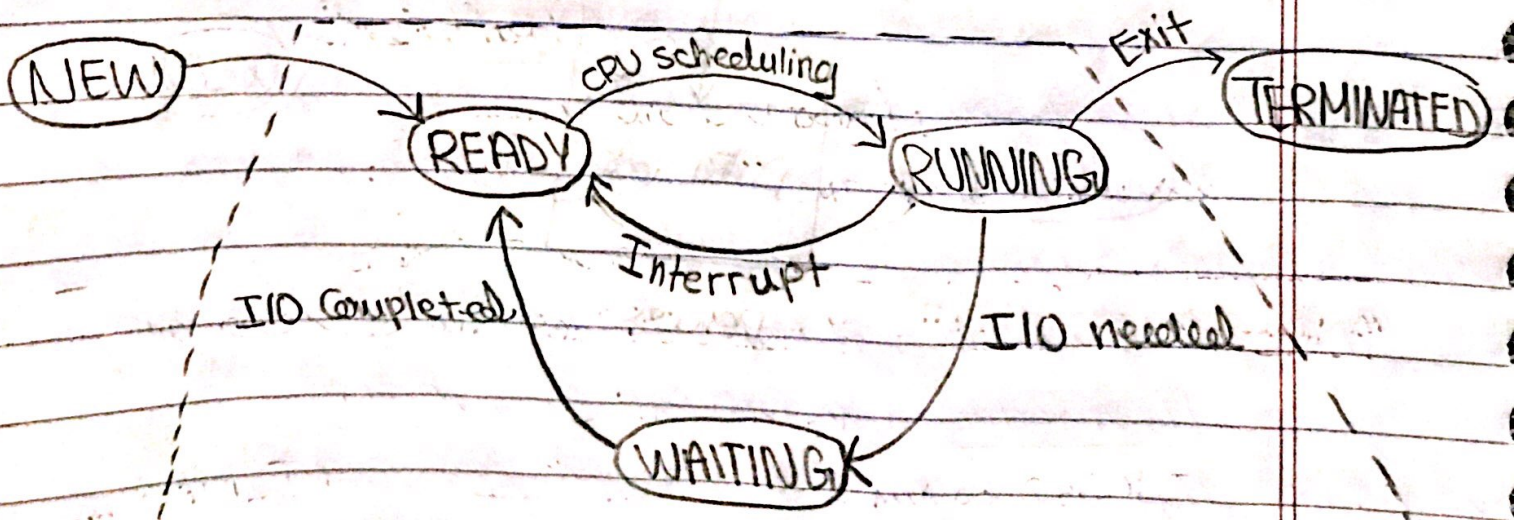
Job Queue "Pool"
newly Created Process

(2) READY: The process is Loaded into Memory & ready to run and have the CPU. The process is in READY queue waiting to have the CPU to run.

(3) RUNNING: The process is having the CPU, & execution.

(4) WAITING: waiting for some event to happen, typically, waiting for I/O.

(5) TERMINATED: The process finishes execution.



Major process state.

Lecture #8

Feb. 26. 2018
Monday

Where data about the process is stored?!

→ Answer → Process Control Block (PCB):
It's a kind of a data structure, generally it's a table.

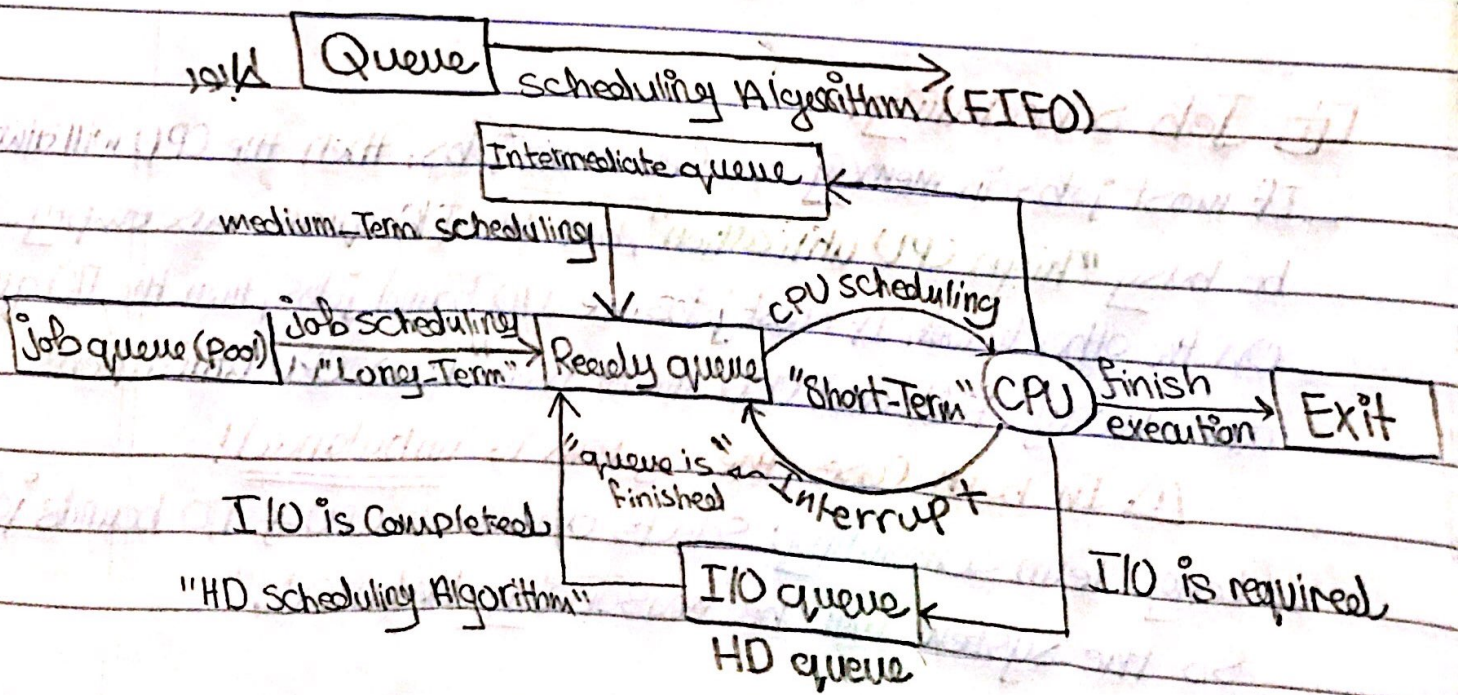
Data?!

(1) process ID.
(2) Process State.
(3) program Counter (PC).
(4) Registers.
(5) Scheduling Info.
(6) Memory Info (Base & limit Registers).
(7) Accounting Info.

PCB

Process ID
Process state
program Counter
⋮

Scheduling & System Queues.



Long-Term "job" scheduling:

It's selecting a job from the job queue to be admitted to memory and in term it's added to the READY queue. This selection "Interupt" is Invoked "Called", this invoked (Seconds, Minutes). That is, the OS has enough time to decide carefully which job to fetch. "Long-Term"

Short-Term Scheduling:

It's selecting a process "job" from READY queue to be given to CPU to RUN.
(milliseconds, microseconds, Nanoseconds), Therefore it should be very fast. "short-Term"

→ Definition → Degree of multi-programming: is the # of jobs in the memory "READY queue". Therefore, Long-Term scheduling controls the degree of multi-programming.

Job scheduling:

- If most jobs in memory are CPU bound jobs, then the CPU will always be busy "high CPU utilization", but the I/O queues are empty.
- On the other hand, If most jobs are I/O bound jobs, then the I/O queues are always full and CPU almost free "Low CPU utilization"

⚠ In both cases the system is unbalanced!

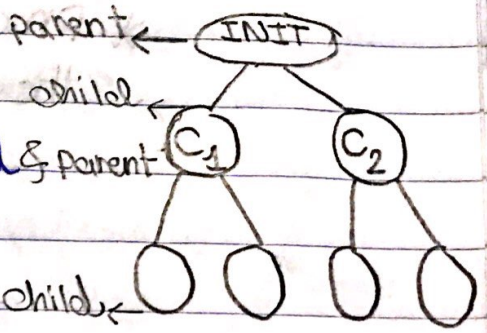
- Long-Term scheduling: selects a mix of CPU & I/O bound jobs So the System will be reasonably balanced.

Lecture #9

Feb. 28. 2018
Wednesday

Process Creation:

* parent process creates children processes, which, in turn create other processes, forming a Tree of processes.



→ Resource Sharing:-

(1) parent & children share all resources.

(2) children share subset of parent's resources.

(3) Parent & child share no resources. "parent & children Compete for All resources"

→ Execution:-

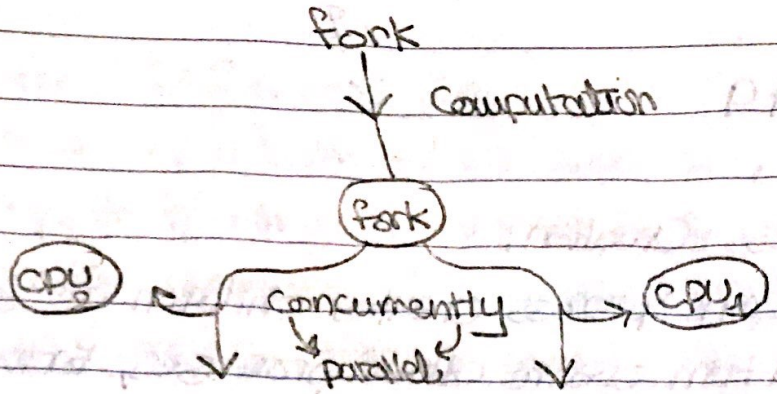
- Parent & children execute Concurrently.
- Parent waits until children terminate.

→ Address Space:- "Allocation of process in Memory"

- child duplicate of parent.
- child has a program loaded into it.

→ Unix examples:

- Fork System call creates new process.
- execve system call used after a fork to replace the process' memory space.



Process termination :-

* Process executes last statement & asks the operating System to delete it 'exit'
  - output data from child to parent 'via fork'
  - process' resources are deallocated by OS. ← when a process is killed, all of its resources are deallocated.

* parent may terminate execution of children processes 'Abort'
  - task assigned to child is no longer required.
  - child has exceeded allocated resources.
  - Parent is exiting.
    # OS doesn't allow child to continue if its parent terminates.
    # Cascading Termination: If the parent process ends 'exit' all children & subchildren are exited.

Cooperating processes :-

Concurrent processes are either :

(1) Independant process : Cannot affect or be affected by the execution of another process.

(2) Cooperating process : Can affect or be affected by the execution of another process.

* Advantages of process cooperation :
  - Information Sharing.
  - Computation Speed-up.
  - Modularity.
  - Convenience.

→ Producer-Consumer Problem

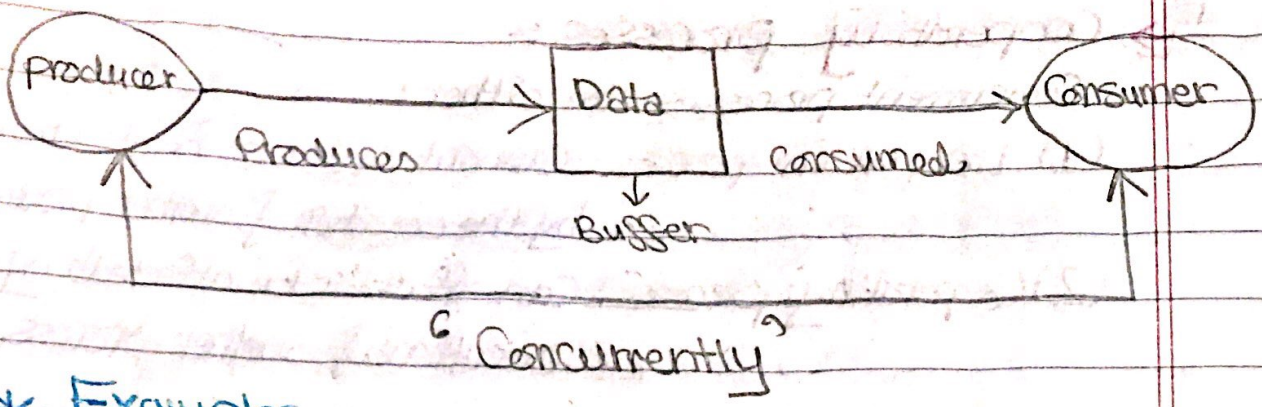
↳ example on Concurrent process

* talking about Concurrency requires :
  - Cooperating among processes
    - (i.e Communication among processes)
  - Synchronization of processes action.

* To illustrate the idea of cooperating process consider the producer-consumer problem :

Producer process produces information.

Consumer process consumes this information.



"Concurrently"

* Examples:-
  - print program produces characters Consumed by the Printer.
  - Compiler produces assembly Code, Consumed by the assembler.
  - assembler produces machine Language Code Consumed by the Loader.

Lecture # 10

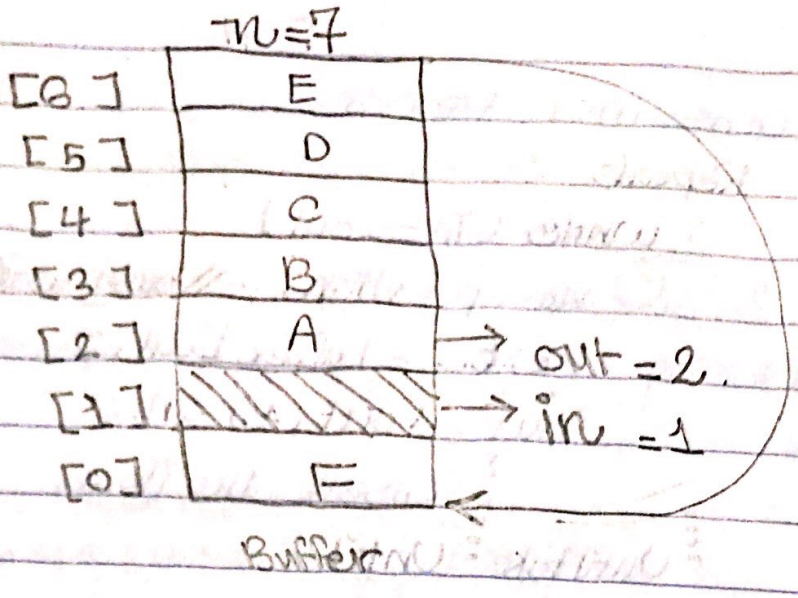
March 3, 2018
Saturday

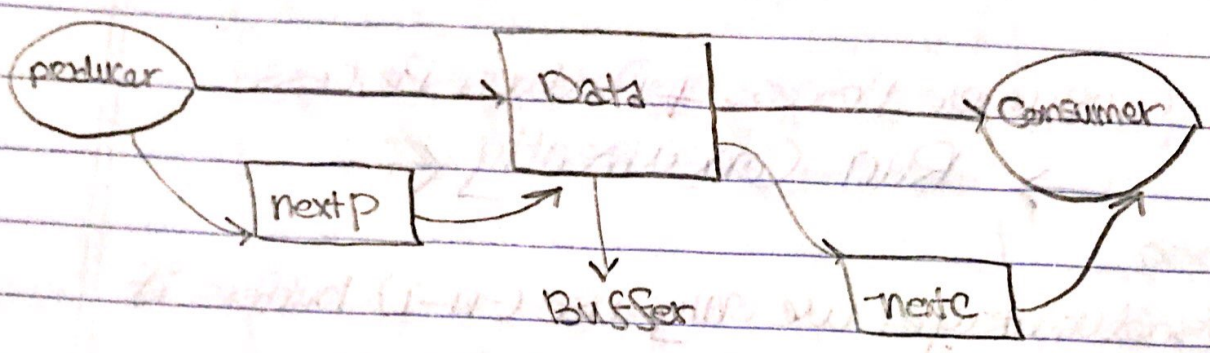
* Data Structure Required:

```
Const int n ; //Size of Buffer.
type item; // item = char, bit, word.
var int buffer [n];
int in; // index where we add items in the buffer.
int out; // index where we take items from the buffer.
item nextP; // next produced item.
item nextC; // next Consumed item.
```

⚠ we will use the Circular Buffer idea in the implementation.

32





* Buffer is Full: $(in+1)\%n = out$.

Circular Buffer

* Buffer is Empty: $in = out$.

* Producer Process:

```
Repeat
    produce an item in nextP.
    while (in+1)%n == out
        no-operation;          // busy waiting
    buffer[in] = nextP;
    in = (in+1)%n;
until False
```

* Consumer process:

```
Repeat
    while (in == out)
        no operation;   //busy waiting
    nextc = buffer[out];
    out = (out + 1) % n;
    Consume the item;
Until False
```

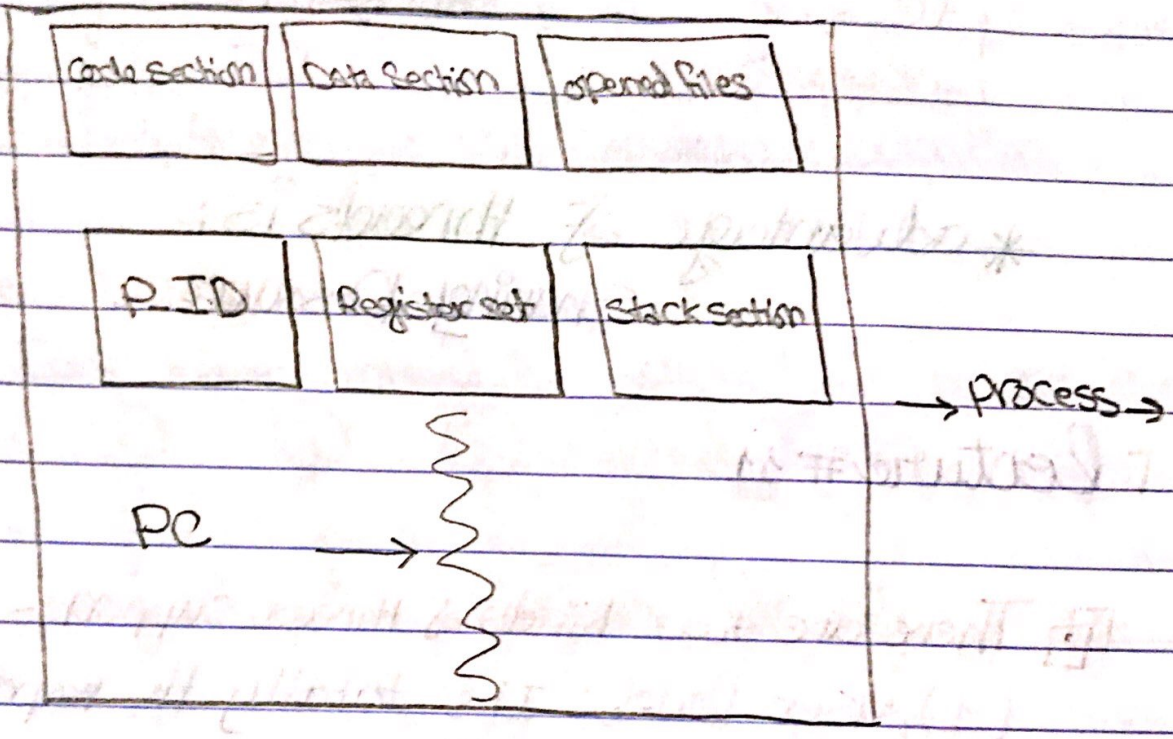
Consumer Process + Producer Process
→ Run Concurrently ←

* Disadvantage: we only use (n-1) buffers if we have n buffers.

Threads:

Process (Heavy Weight Process)

Code section	Process ID
Stack section	Registers set
Data section	Program Counter

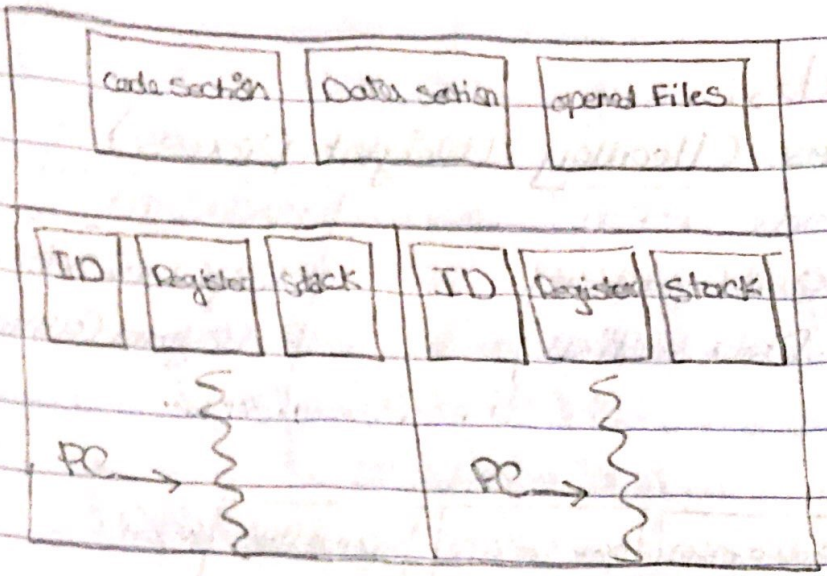


Thread (Light Weight Process) Contains:-

- Program Counter.
- Registers set.
- Stack section.

→ All peer threads; share:

- Code section.
- Data section.
- I/O Resources.



*advantage of threads is:
Sharing Resources.

lecture #11

March 5, 2018
Monday

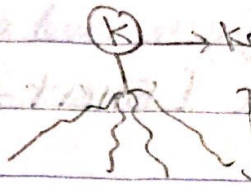
There are two kinds of thread support:-
(1), user level: It's totally the responsibility of the user (very Complex & difficult)
at
(2) at kernel level: most OS support this kind of threads.

* Relationship between user threads & kernel threads:
→ Think about user threads as processes (programs)
&
→ Think about Kernel threads as CPUs

* We have → multi-threads.
→ multi-CPUs.

(1) Many-To-One:

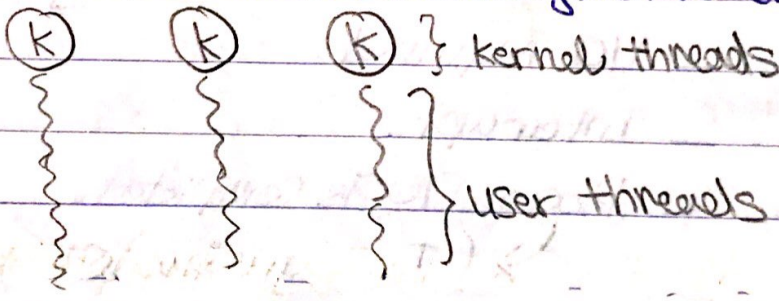
Many user threads are mapped to one kernel thread



⚠ Disadvantage: No concurrent execution.

(2) One-To-One

Each kernel thread is assigned to one user thread.



∴ Main advantage: This allows concurrent execution.

⚠ Disadvantage: we need enough kernel threads

(3) Many-To-Many:

Many user threads are mapped to an equal or less number of kernel threads.