

Module 1.2.1 OS

- Introduction tot he O/S:
- Intro to linux

Silberschatz Galvin book for O/S :

OS: Controls the hardware and co ordinates its use among the various application programs for the various users!

The layers in system:

- Application
- OS
- Computer Orgnaisation
- VLSI - Very large-scale integration (VLSI) is the process of creating an integrated circuit (IC) by combining millions of MOS transistors onto a single chip
- Transistors - A transistor is a device that regulates current or voltage flow and acts as a switch or gate for electronic signals. Transistors consist of three layers of a semiconductor material, each capable of carrying a current.

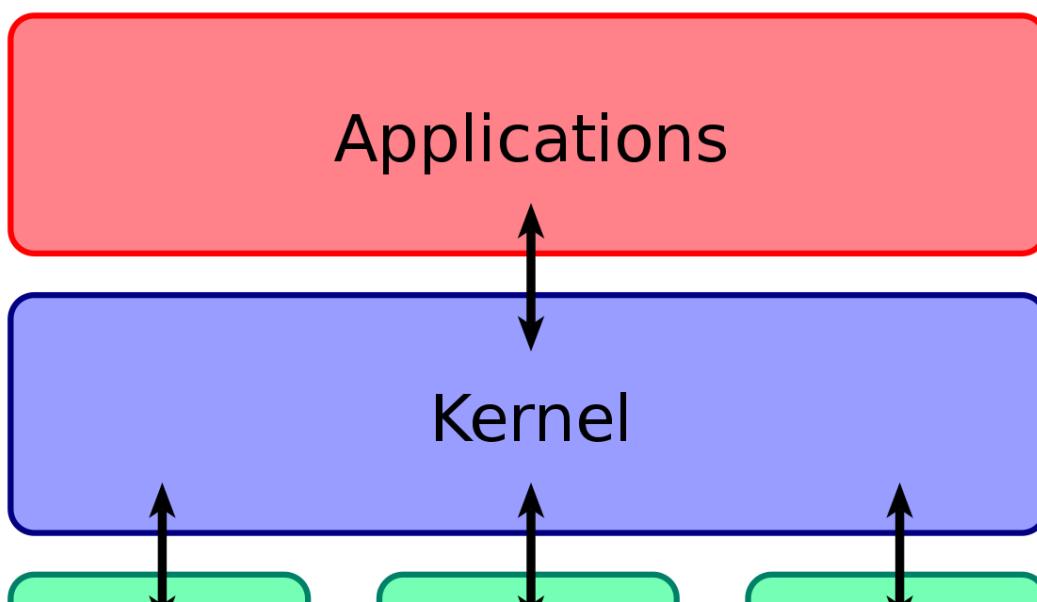
HARDWARE ABSTRACTION :

RESOURCE MANAGEMENT :

Kernal : The kernel is the essential center of a computer operating system (OS). It is the core that provides basic services for all other parts of the OS. It is the main layer between the OS and hardware, and it helps with process and memory management, file systems, device control and networking.

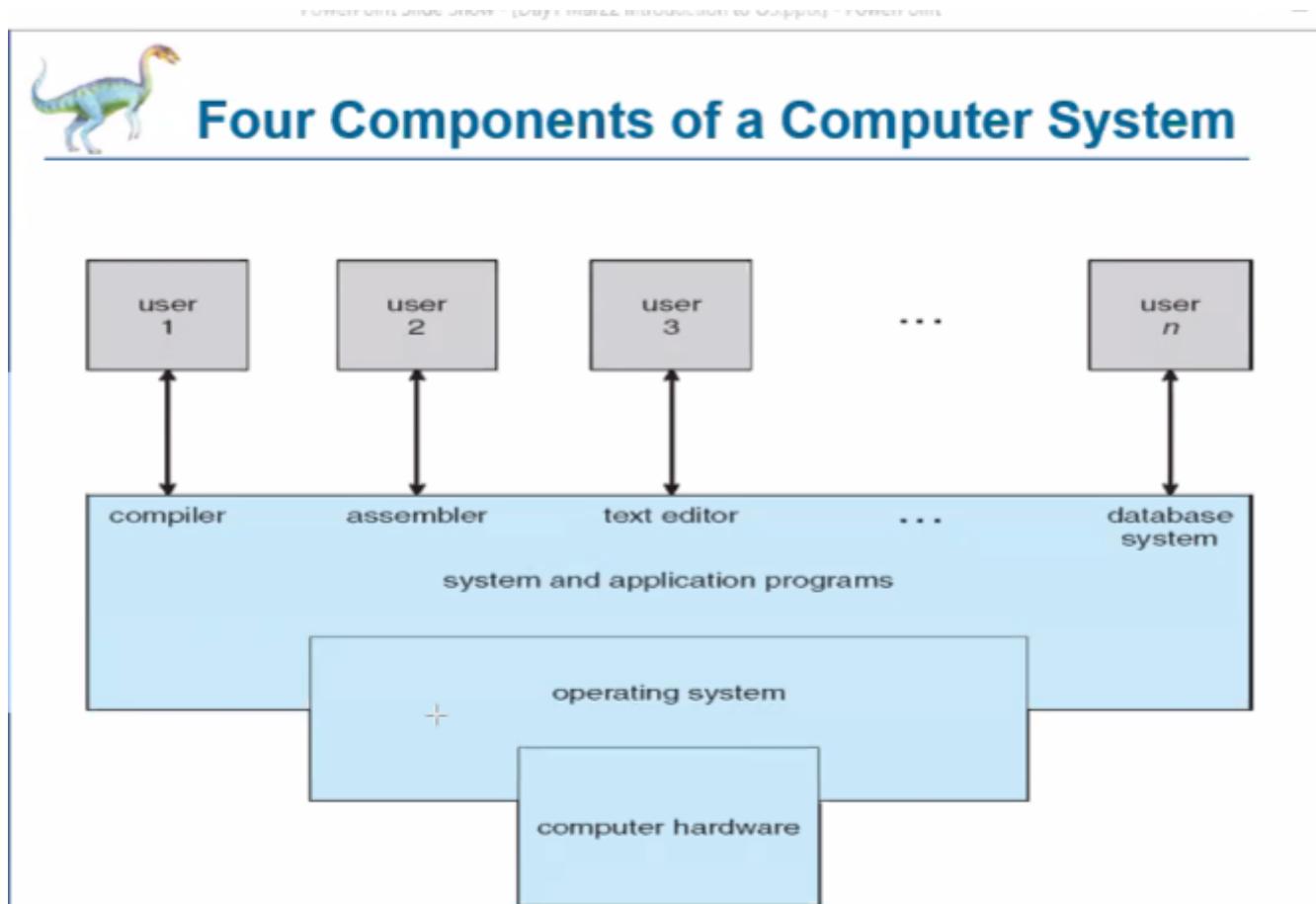
It's the program which runs all the time in computer:

Computer program and the core of OS with the complete control over everthing in the system.



CPU**Memory****Devices**

OS is also called as Resource allocator OR Control Program:



System Boot:

Booting is the process of starting a computer. It can be initiated by hardware such as a button press or by a software command. After it is switched on, a CPU has no software in its main memory, so some processes must load software into memory before execution. This may be done by hardware or firmware in the CPU or by a separate processor in the computer system.

- When switch on system : bootstrap program will be loaded
- Volatile(RAM) and (ROM)Non-Volatile memory(permanent memory) our system has
-

TYPES OF MEMORY IN COMPUTER :

Von-Neumann Model :

PROGRAM VS PROCESS :

program is the set of instructions

When the program is loaded into memory and waiting for the execution is the process!



Operating System Management Tasks

1. **Process management** which involves putting the tasks into order and pairing them into manageable size before they go to the CPU.
2. **Memory management** which coordinates data to and from RAM (random-access memory) and determines the necessity for virtual memory.
3. **Device management** provides an interface between connected devices.
4. **Storage management** which directs permanent data storage.
5. **An application** that allows standard communication between software and your computer.
6. **The user interface** allows you to communicate with your computer.

What are the different operating systems for mobile phones?

Android (Google)

iOS (Apple)

Bada (Samsung)

Blackberry OS (Research in Motion)

Windows OS (Microsoft)

Symbian OS (Nokia)

Tizen (Samsung)

A real-time operating system (RTOS) is an operating system with two key features: predictability and determinism. In an RTOS, repeated tasks are performed within a tight time boundary, while in a general-purpose operating system, this is not necessarily so.

OS From Utube :

- Basic concepts:
 - Types of OS, System calls, Kernel, Process diagram
- Process Scheduling (Easy and very important topic)
 - FIFO, SJF, PR, RoundRobin Algorithms
- Process Synchronisation :
 - Semaphore
- Deadlock adn thread
 - Bankers alogortihms
- Memory management
 - Paging
 - segmentation
 - fragmentation
 - Virtual memory
 - Page replacement algorithm
- Disk Scheduling
 - sCAN, cscan aLGO,fcfs
- UNIX Commands
 - ls, mkdir, chmod, cd etc.
 - Open System calls
- File Management and security
 - attacks and it's types

An operating system is a piece of software that manages all the resources of a computer system, both hardware and software, and provides an environment in which the user can execute his/her programs in a convenient and efficient manner by hiding underlying complexities of the hardware and acting as a resource manager.

An Operating System can be defined as an interface between user and hardware. It is responsible for the execution of all the processes, Resource Allocation, CPU management, File Management and many other tasks.

The purpose of an operating system is to provide an environment in which a user can execute programs in convenient and efficient manner.

What does an Operating system do?

1. Process Management
2. Process Synchronization
3. Resource Management
4. Memory Management
5. Ram Management
6. CPU Scheduling
7. File Management

shorthand : PP MRF CS

Types of OS :

- Single processing OS

Eg. MS-DOS

- Batch eg ATLAS

- similar kinds of job given to the pc for the execution
- one batch at one time
- its very old system
- no longer used now as the complexities raised
- Job is given to the operator, he converts that into the batches and B1 B2 B3 and it will be given to the CPU and CPU will be idle this was the major disadvantage of this system
- CPU will execute the next task only after the completion of the one job
- Monitors were introduced to replace the opeartos
- CPu utilisation was not full
- Process starvation problem was still there.

Fortran : is a general-purpose, compiled imperative programming language that is especially suited to numeric computation and scientific computing.

First appeared: 1957; 65 years ago

IBM 7090/94 IBSYS

IBSYS itself is a resident monitor program, that reads control card images placed between the decks of program and data cards of individual jobs. An IBSYS control card[a] begins with a "\$" in column 1, immediately followed by a Control Name that selects the various IBSYS utility programs needed to set up and run the job. These card deck images are usually read from magnetic tapes prepared offline, not directly from the card reader.

- Multi Programmed eg THE

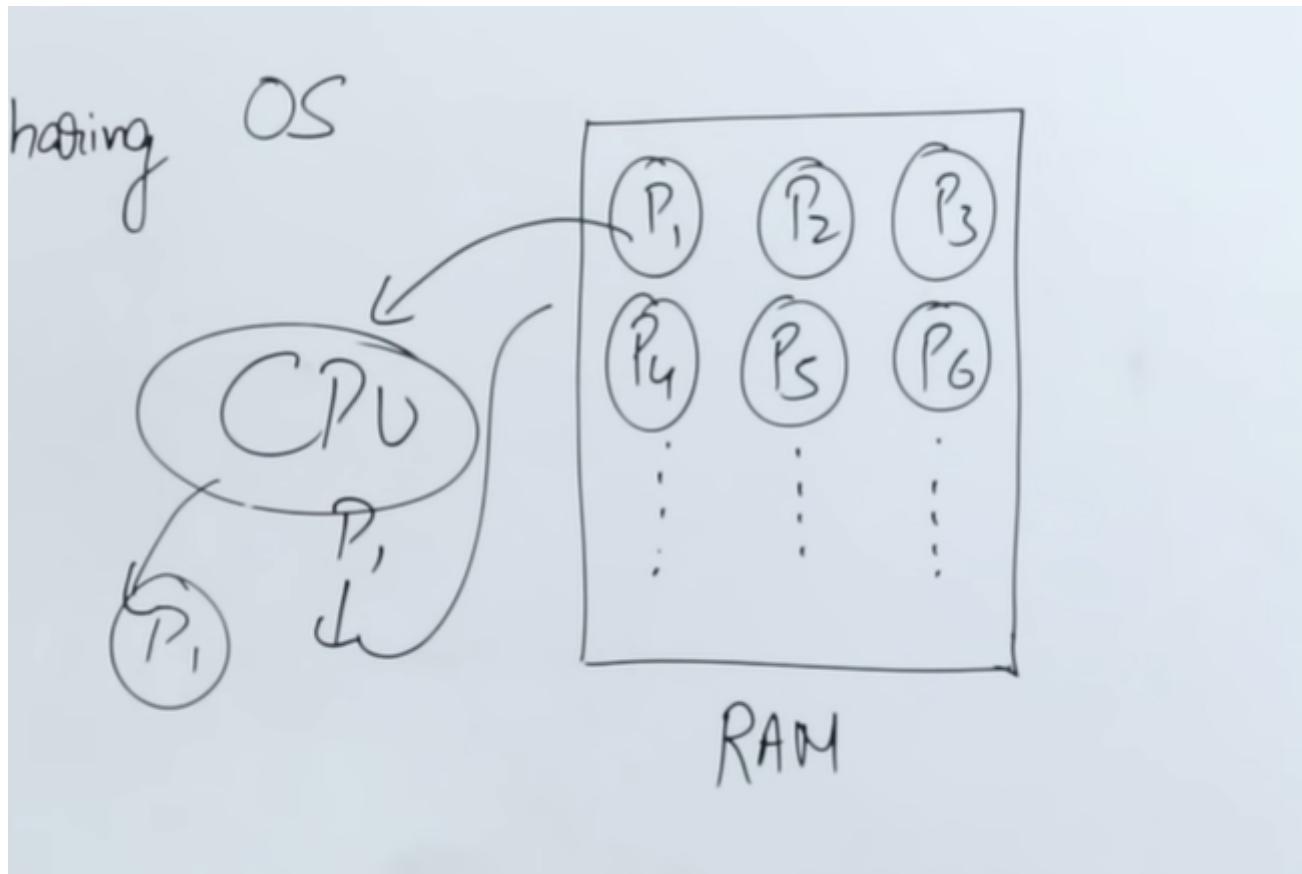
- CPU ideal time was reduced with this process
- Non preemptive - will finish the once process only after that will go to the next one
- Multiple processes were brought into the ram and then P sent to CPU for processing and CPU will execute one program at a time till its end until and unless it ask for any i/p or o/p
- came ebefore multitasking

- Multi tasking/ time sharing eg CTSS

- Preemptive - will switch on fixed time intervals
- In this case the CPU will go to every process without ending the complete process, here CPU has set the time that it will give the fixed amount of time to each and everyt process and even if

it didn't get completed then also it will go to the next process.

- In this case the major advantage is that the last process will have to wait less as compared with the multiprogramming OS and its turn will come early and after that it will go to the next process.
- Responsiveness is high here
- After specific time quantum (**Applicable in Round Robin Algo**) the process will go back to the ready queue and wait in the ready queue.
- **Context switching** is applicable here which means the process after specific time quantum will back in the queue

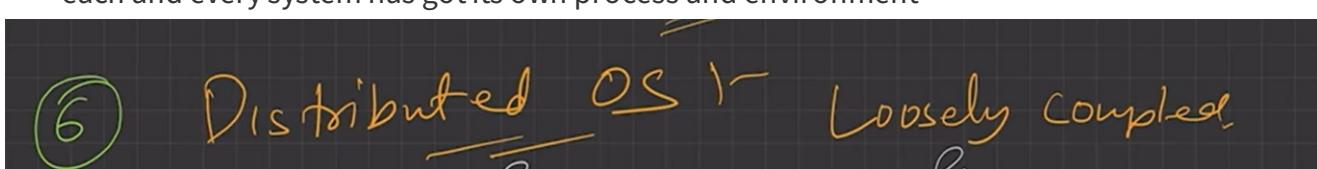


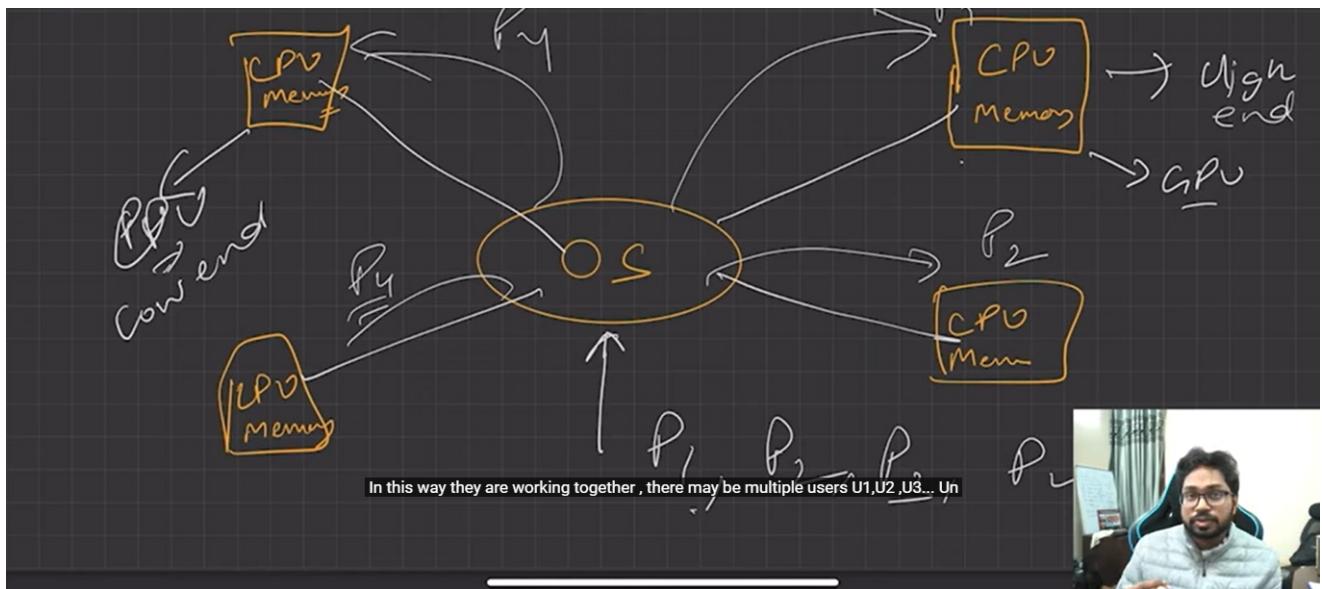
- MultiProcessing eg Windows

No of CPU > 1

LET'S ONE CPU HAS 4 CORE then 8 cores is like 2 CPU so it's multiprocessor OS.

- Real time OS
 - Live system
 - time is very important
 - hard real time - no delay accepted
 - soft real time - small delay is accepted
- Distributed
 - Geographically separated but connected with the network
 - each and every system has got its own process and environment
-





- Losslessly coupled autonomous systems.
- Clustered
 - its similar to the distributed but its clustered!
 - its power increases
 - it acts as super computer
- Embedded
 - works on a fixed system
 - e.g. washing machine, microwave etc. they perform fixed functions

program: Set of instruction stored in file on

process : Program under execution is called process in RAM

thread : light weight process or small part of process

Process states:

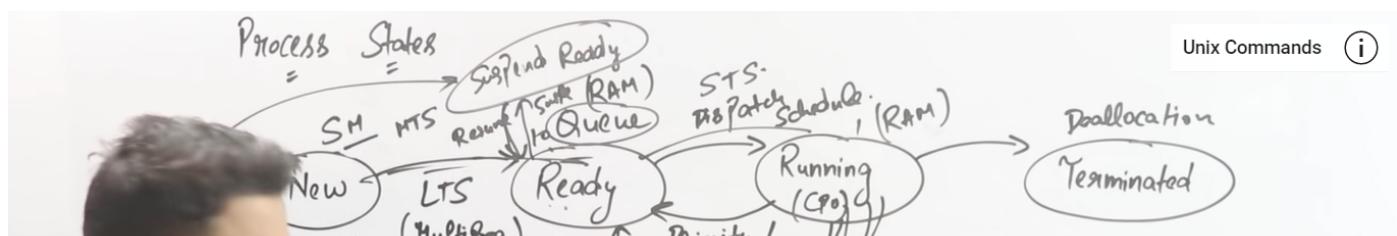
new: Start of the program eg.g. program is created and stored on ROM

ready (queue) : various processes send to the RAM by LTS

Running: Here the program is still in the ram but now its sent to CPU for the operation and its done by the STS and if ends the program then its called as multiprogramming and if sent back after time quantum then its called as an multiprogramming and the program here is sent back to the ready queue and the next program is taken for the execution in cpu.

Terminated: and if the program is completed then it goes to this stage and deallocation is done in this stage all the resources allocated to this stage are taken back from the program.

wait/ block :if the program needs any kind of i/p or o/p then it goes for the wait block





PS command is used to check the process stage in linux OS

Important linux commands:

chmod ugo=r file_name.ext => used to give read acces to all the three groups

lseek command:

System calls in OS:

WINDOWS HAS 700+ SYSTEM CALLS

Linux has 300+ System calls

System calls are used to interact with the hardware thru kernel

API are used in programming languages such as printf(), to print the o/p on screen and here it's called as API's.

File related :

open(), Read(), Write(), close(), CreateFile()

Device Related:

Read, Write, Reposition, ioctl, fcrtl

Information calls:

get pid(process ID), PPid (Parent process ID), attributes, get system time and data, to get teh metadata ;

Process control:

Load, Execute, Abort, Fork, wait, signal, allocate etc.

Communication:

pipe(), create/delete connection,

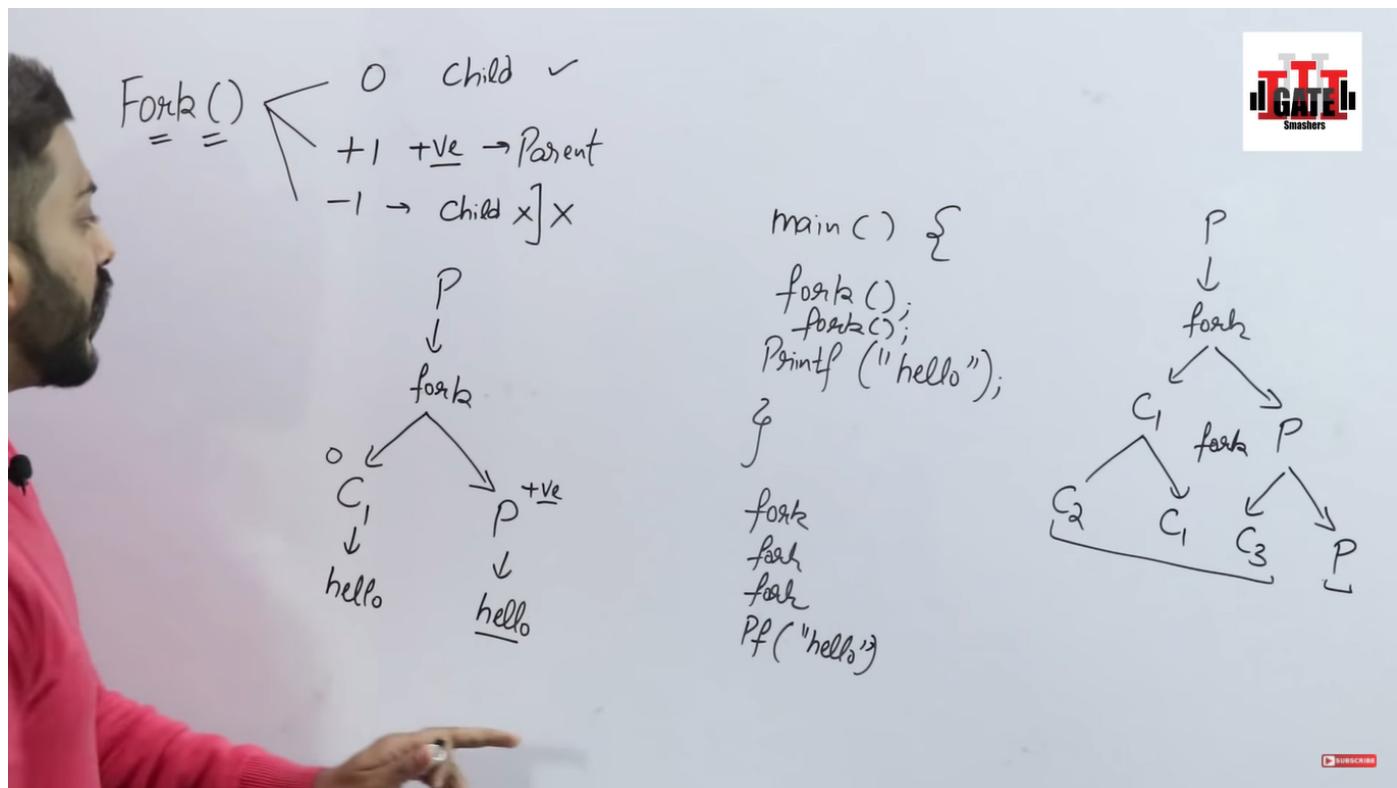
Fork system call;

fork creates child to perform some task

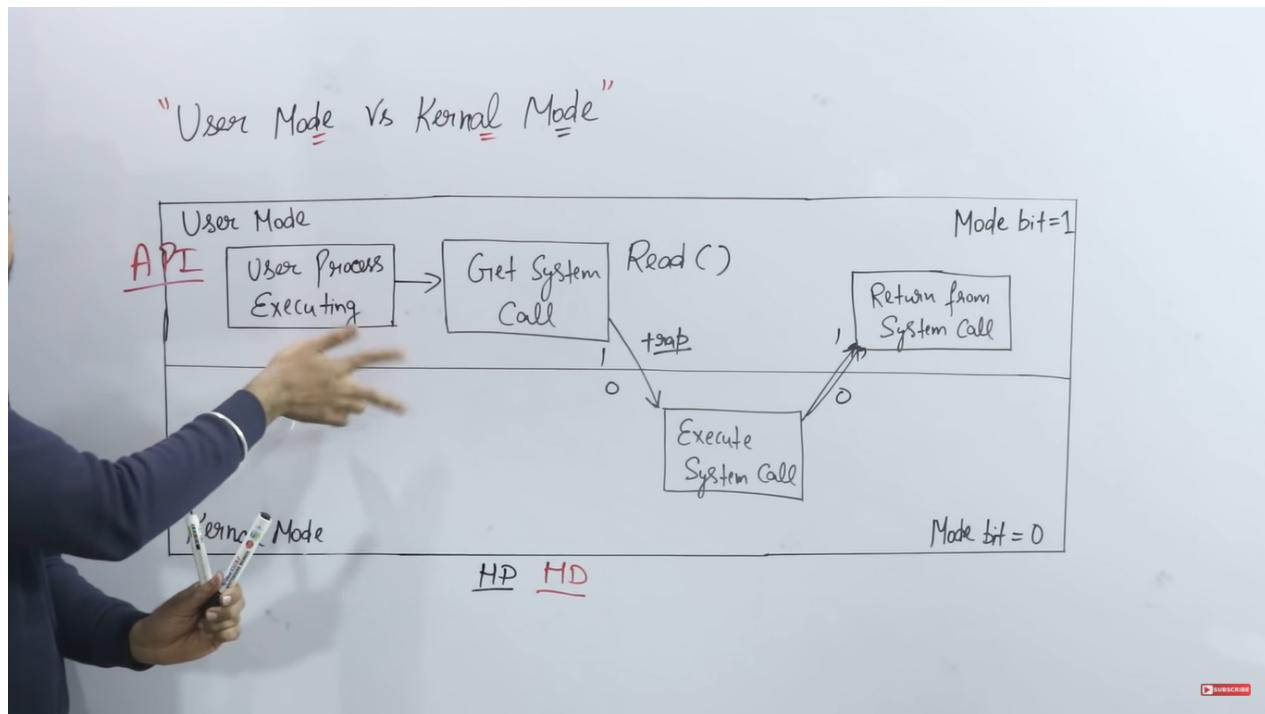
CHILD PROCESS CREATED

+1 is parent process

-1 child process doesn't created

Total number of child process generated = $2^{**n} - 1$

here n is the no of times fork has created



Difference between process and threads!

Process

- 1) System Calls involved in Process
- 2) OS treats different processes differently
- 3) Different process have different Copies of Data, files, Code
- 4) Context switching is slower
- 5) Blocking a process will not block another

A hand in a red sleeve points to the fifth item.

Threads (User Level)

- 1) There is no system call involved
- 2) All User level threads treated as single task for OS
- 3) Threads share same copy of code and data
- 4) Context switching is faster
- 5) Blocking a thread will block entire process
- 6) Interdependent

YouTube subscribe button

USER LEVEL THREADS VS KERNEL LEVEL THREADS:

"User level Thread"	"Kernel Level Thread"
 <p>User level threads are managed by User level library</p> <p>User level threads are typically fast</p> <p>Context Switching is faster</p> <p>If one user level threads perform blocking operation then entire process get blocked</p> <p>$\frac{\text{Process}}{\text{CT}} > \text{KLT} > \text{ULT}$</p>	<p>Kernel level threads are managed by OS System Calls</p> <p>Kernel level threads are slower than User level</p> <p>Context Switching is slower</p> <p>If one kernel level thread blocked, No affect on others.</p>

YouTube subscribe button

Process scheduling algorithms:

CPU Scheduling algorithms:

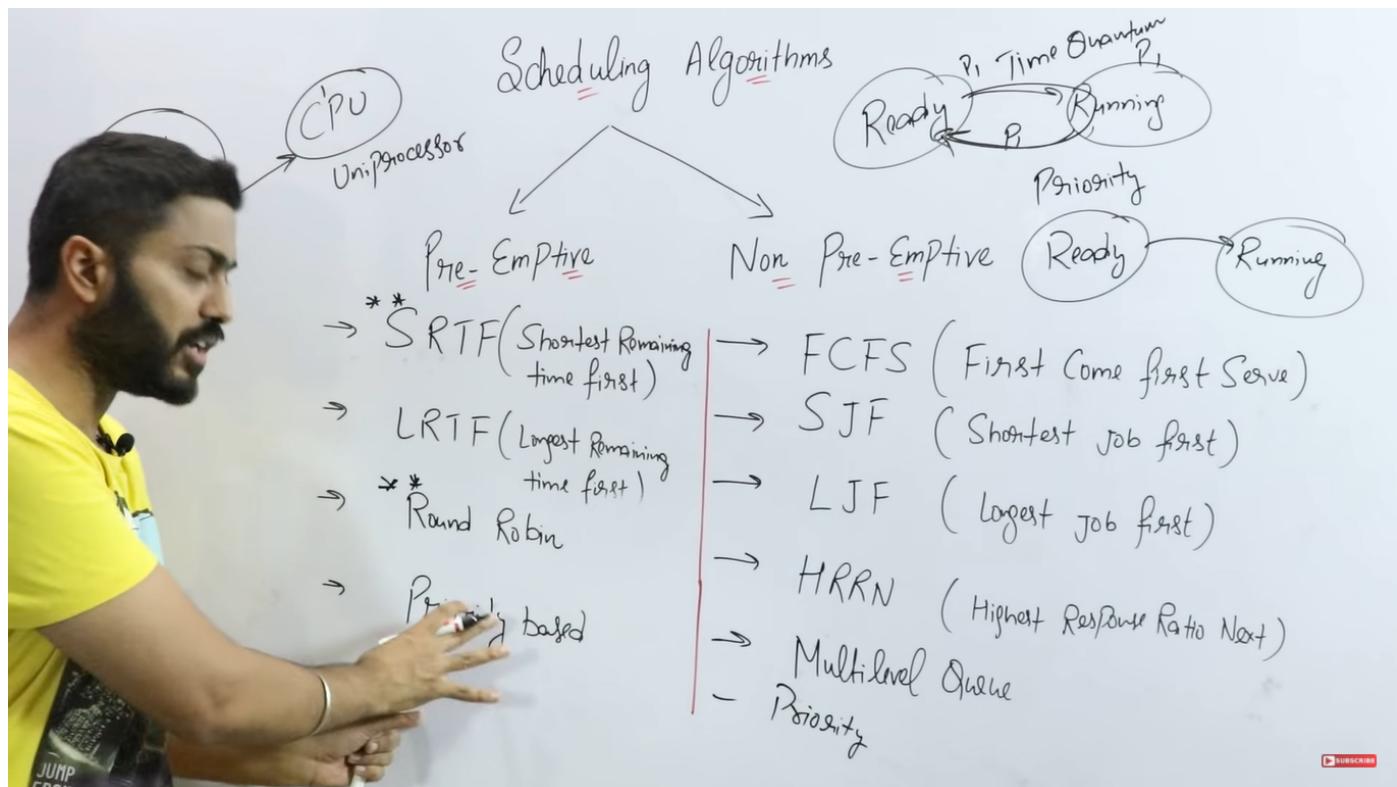
Non-Premptive:

- FCFS
- SJF
- LRF
- HRRN : Highest Response Ratio First

- Multi level Queue
- Priority

Preemptive Alogos:

- SRTF
- LRTF
- Round Robin
- Priority Based



L-2.2: What is Arrival, Burst, Completion, Turnaround, Waiting and Response time in CPU Sch...

CPU Scheduling

→ Arrival time: The time at which process enter the Ready Queue or State

→ Burst time: Time required by a process to get execute on CPU.

→ Completion time: The time at which process complete its execution.

→ Turn Around time: $\{ \text{Completion time} - \text{Arrival time} \}$

→ Waiting time: $\{ \text{Turn Around time} - \text{Burst time} \}$

→ Response time: $\{ (\text{The time at which a process get CPU first time}) - (\text{Arrival time}) \}$

12 - 11 = 1 hour = 60 minutes

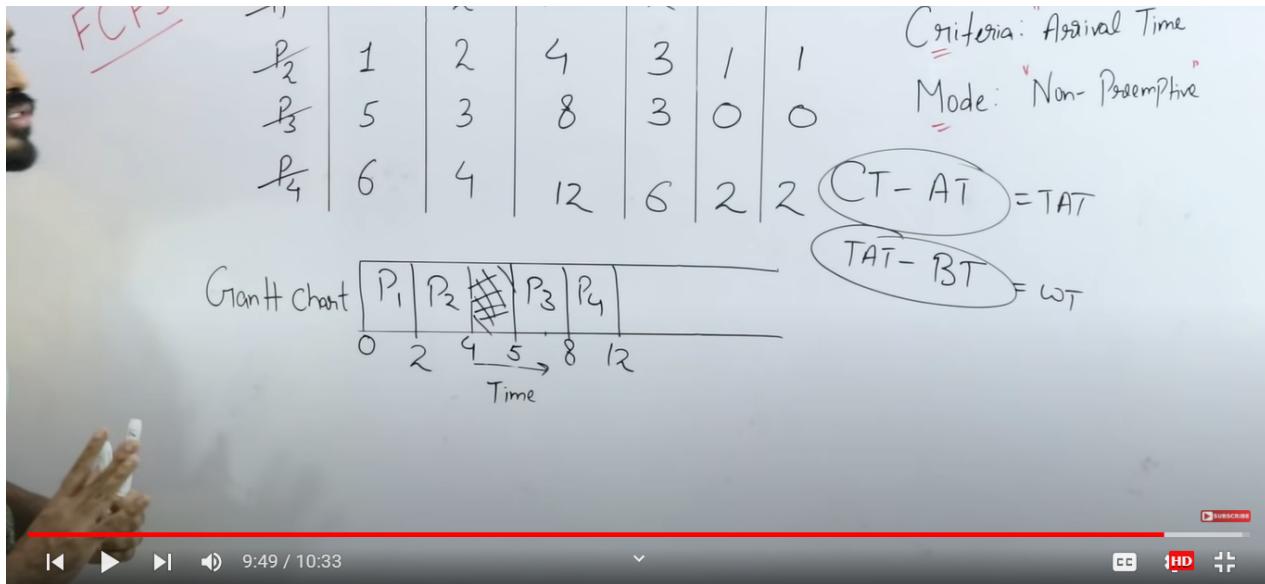
CPU bound
I/O bound

60 minutes - 15 min = 45 minutes

8:30 / 8:58 • Response time >

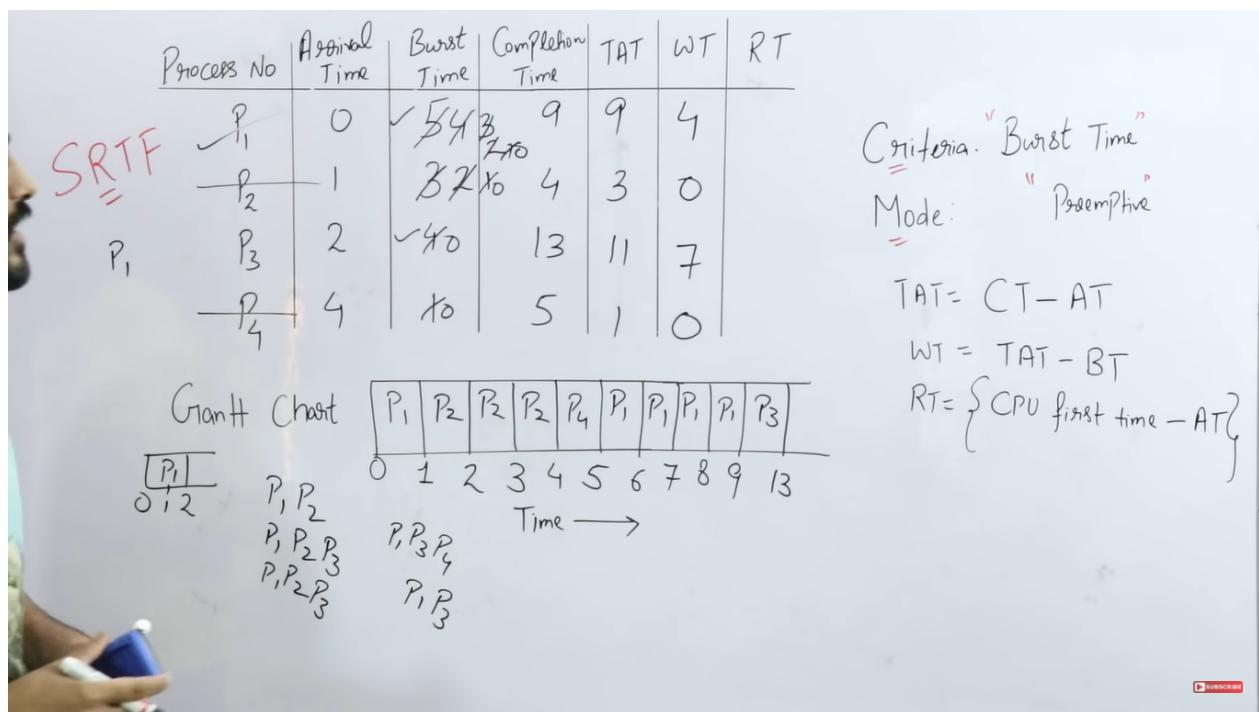
L-2.3: First Come First Serve(FCFS) CPU Scheduling Algorithm with Example

Process No	Arrival Time	Burst Time	Completion Time	TAT	WT	RT
P1	0	2	2	2	0	0



Response time is same as Waiting time in case of non preemptive mode:

SRJF: Shortest Running Job First (It's a preemptive mode but time quantum is not applicable here process with the less burst time is selected)

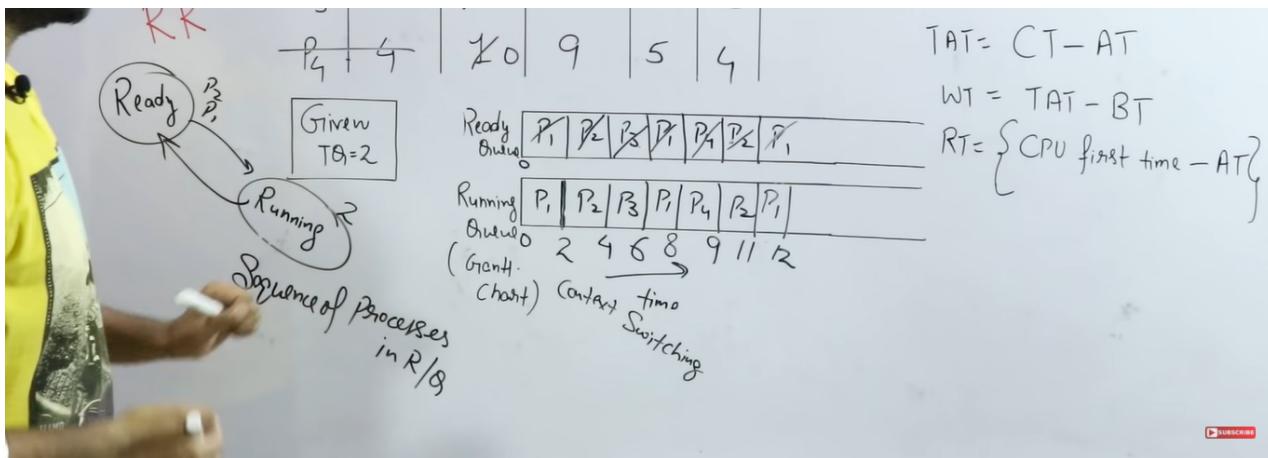


in case of preemptive Response time is very important in above e.g.:

Preemptive means the process after certain time quantum will go back to the ready queue!

Round Robin : TQ is applicable after TQ process will be sent back to the





Note: In above eg check the P₄ CPU scheduling here burst time is less than TQ (2) i.e. 1 so it will take only 1 unit of time for execution and will take the next process for execution

Context Switching : SAVE THE RUNNING PROCESS and sent back to the ready queue after TQ/ This can be calculated by the no of vertical line between the process on the gantt chart!

Pre-emptive Priority Scheduling Algorithm with Example | Operating System

In this case value of priority is checked! first check is arrival time and then run it for 1 unit time and check the priority after every 1 unit!

The table shows the results of Pre-emptive Priority Scheduling for processes P₁ through P₄:

Priority	Process No	Arrival Time	Burst Time	Completion Time	TAT	WT
10	P ₁	0	5	12	12	7
20	P ₂	1	3	8	7	3
30	P ₃	2	4	12	2	0
40	P ₄	4	10	5	1	0

Annotations on the table:

- Higher the no. higher the Priority
- Priority Scheduling
- Criteria: "Priority"
- Mode: "Preemptive"
- $TAT = CT - AT$
- $WT = TAT - BT$

The Gantt chart shows the execution sequence over time units 0 to 12.

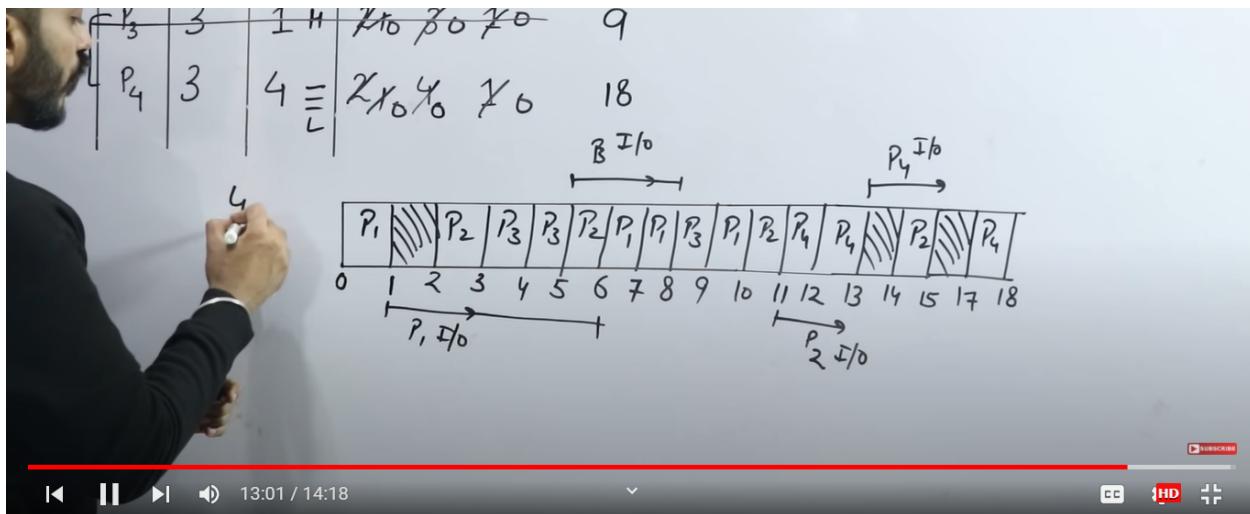
If the priority is same then run it as per the arrival time and if the arrival time is also the same then run it as per the sequence given.

The table shows the mix of CPU and I/O burst times for processes P₁ and P₂:

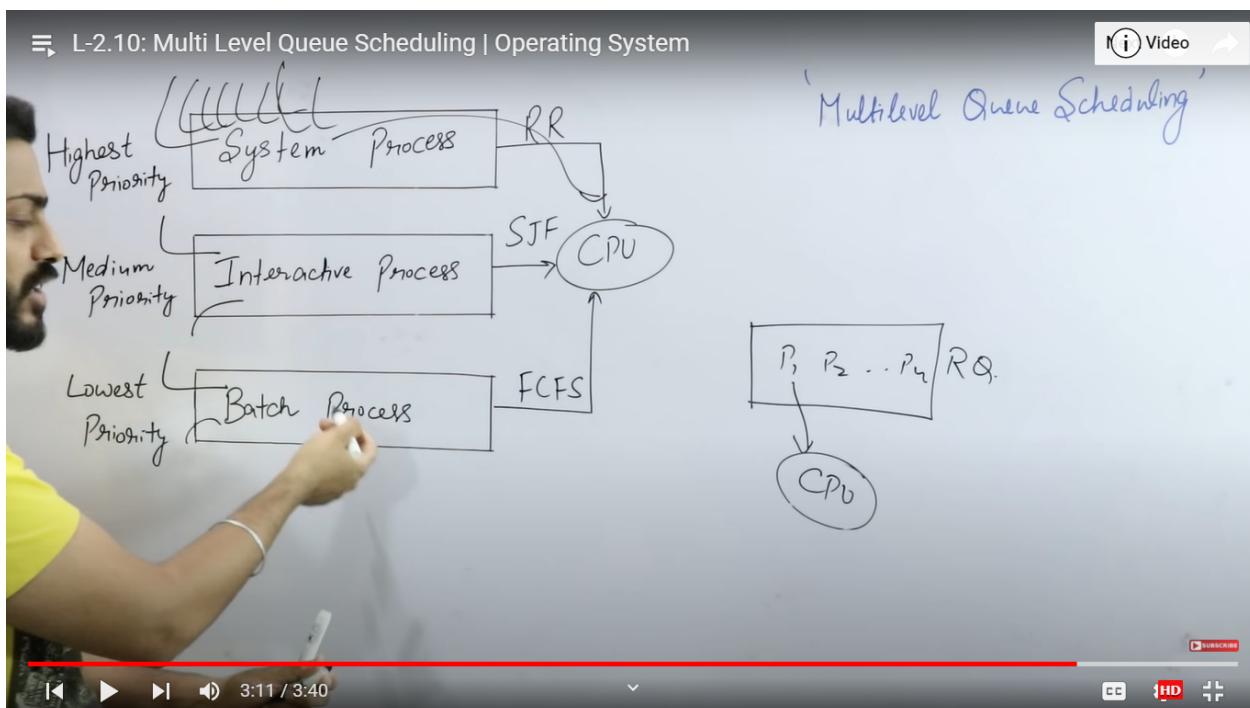
Process	AT	Priority	CPU	I/o	CPU	CT
P ₁	0	2	10	30	20	10
P ₂	2	3	20	30	10	15

Annotations on the table:

- Mode: Preemptive
- Criteria: Priority based
- Find CT of P₁, P₂, P₃, P₄



Multi level Queue Scheduling :



Advantages :

- Different ready queues are created as per the real life examples because every process is different
- for every single queue diff algorithm can be set

Disadvantages:

- Starvation : as if high no of system process came then low priority process wont get time for execution

Multilevel Feedback Queue Scheduling | Operating System

Process Synchronisation:

Independent process:

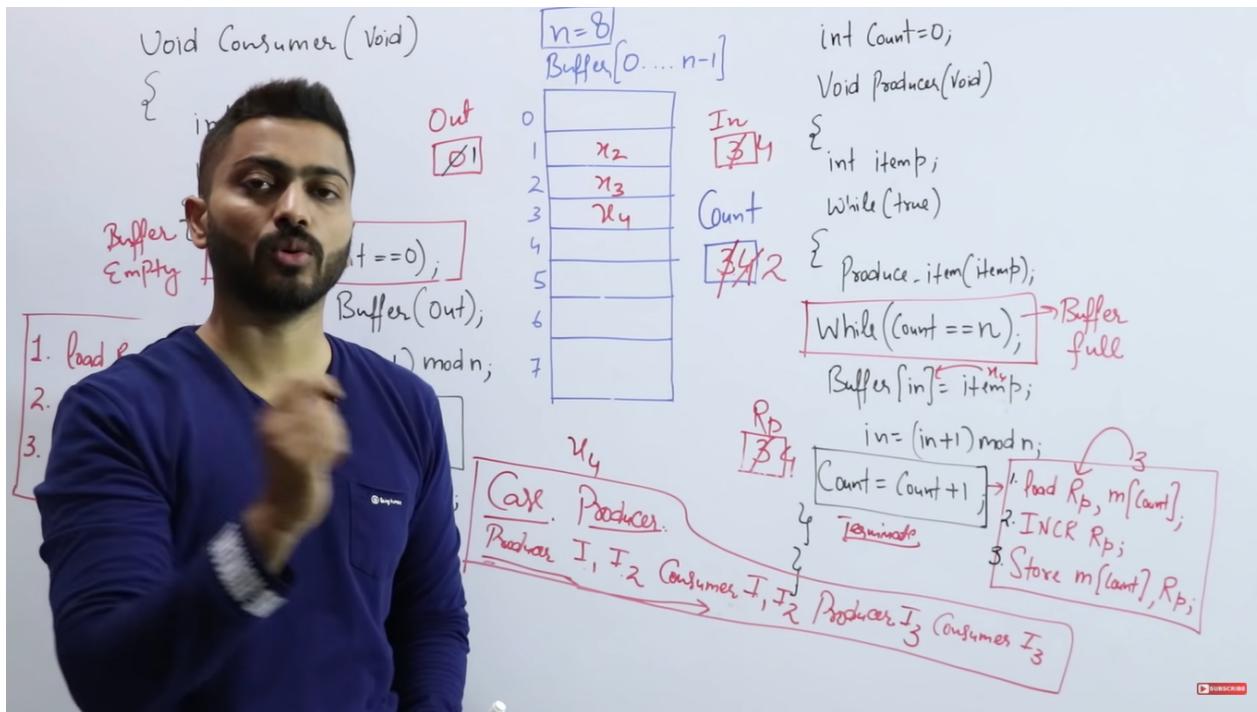
Co operative process:

- Share variable Memory code Resources etc

- Race Condition for using the common resources or data which is responsible for the loss of data and deadlock!
- These co operative processes creates a problem and these needs to be synchronised
- Deadlock situation may occur due to this co operative process

Uniprocessor : Single CPU system

Producer and consumer problem:

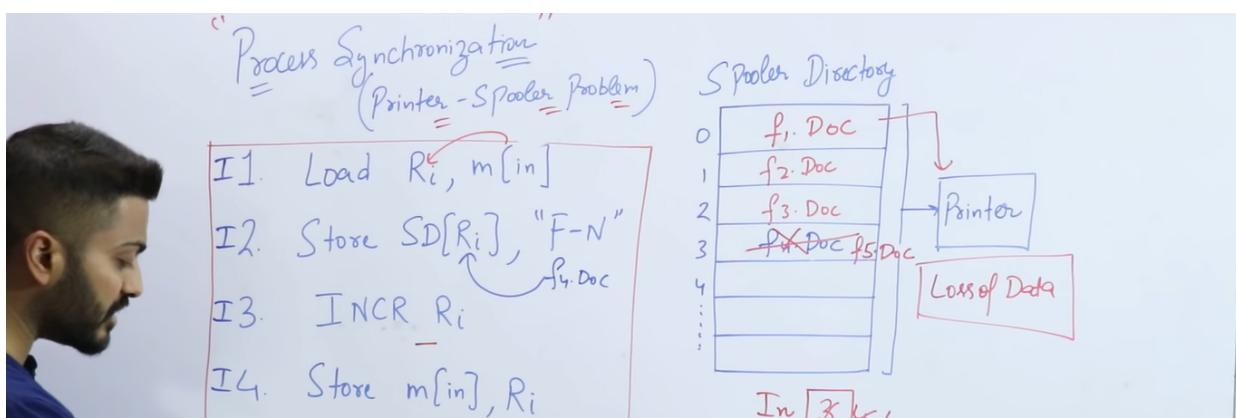


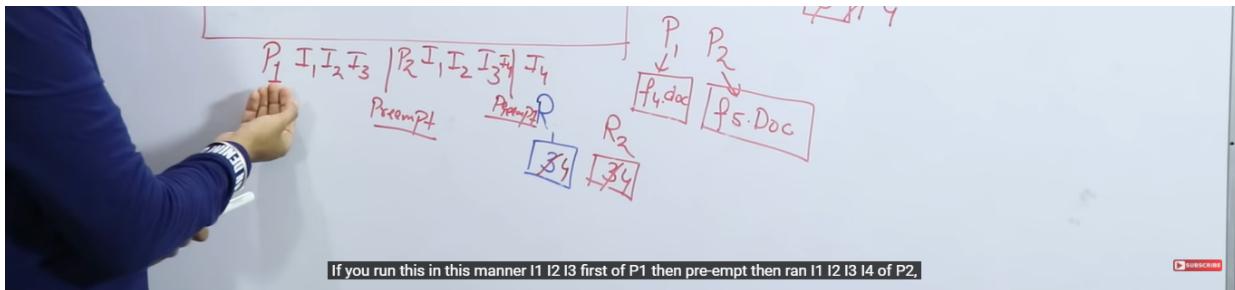
To avoid such issue below methods can be applied:

- Binary Semaphores
- Monitors
- locks
- etc.

Printer Spooler problem:

Printer is basically a very slow device as compared with the CPU, RAM etc. and Spooler is the software in printer which store the no of file sent back to back for printing,





Critical Section:

Place or the portion of the process where shared variables or resources are placed!

These 4 conditions are very important means you can give any solution, any solution.

Bounded Wait:

- When p1 is inside the CS, P2 WILL BE OUT SIDE But this should not happen that p1 is always inside and p2 is not getting a chance to get inside the CS (Critical Section)

No assumption related to Hardware or Speed:

- if synch is made for 32 bit system it should work for 64 bit too.
- All synch code must be universal, irrespective of hardware and speed of the process.

Lock Variable in OS :

- Oldest method :

```

do {
    acquire lock
    CS
    release lock
}

```

Lock != 0

1. While(LOCK == 1); ENTRY CODE
2. LOCK = 1
3. Critical Section
4. LOCK = 0 EXIT CODE

Look here, is there any guarantee that mutual exclusion will always be there?

This process doesn't guarantee the mutual exclusiveness, as if P1 got preemptive immediately after while which without making lock==1 so lock=0 and P2 will enter into the CS and will execute the code and will get out with lock=0 and now when P1 will again come back so it will take lock==1 and enter the CS but by that time it's P2 who is already executed from the CS.

Test and Set instructions in OS:

To avoid the above error, two lines of code were merged into one i.e. test_and_set and now it won't give any error!

Critical Section Solution using "Test_and_Set" Instruction

false

```
while (test_and_set (& lock));
CS P1 P2 T=1 f=0 L=false.
boolean test_and_set (boolean *target)
{
    boolean r = *target;
    *target = TRUE;
    return r;
}
```

1. While(LOCK==1); ENTRY CODE
2. LOCK=1
3. Critical Section P2 ← P1 ←
4. LOCK=0 EXIT CODE

MEX ? P1 P2
1 1
L=<addr>+1
lock=0

It has become true because P1 is already in. Now look, what we have to do set target.

Turn Variable (Strict Alteration): This follows all the conditions to make process synchronise except the process condition:

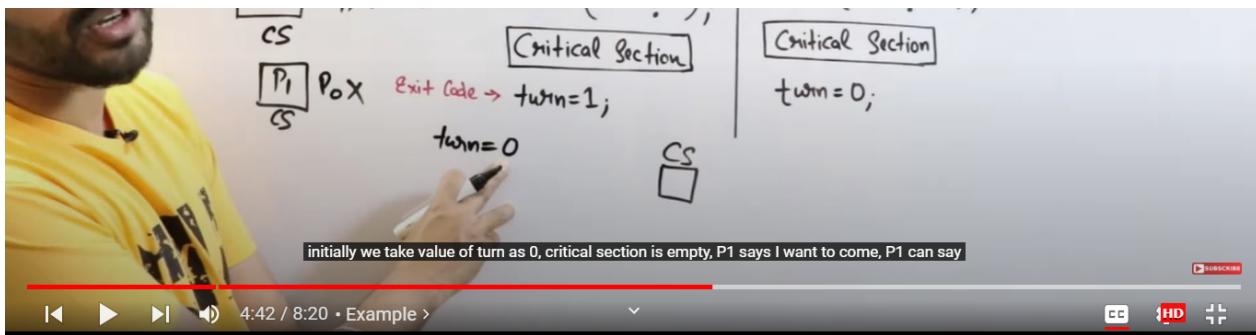
Turn Variable (Strict Alteration)

- * 2 Process Solution
- * Run in User Mode

ME ✓ Program

Process "P0"	Process "P1"
No CS	No CS
While (turn != 0)	While (turn == 1);

Entry Code → While (turn != 0)

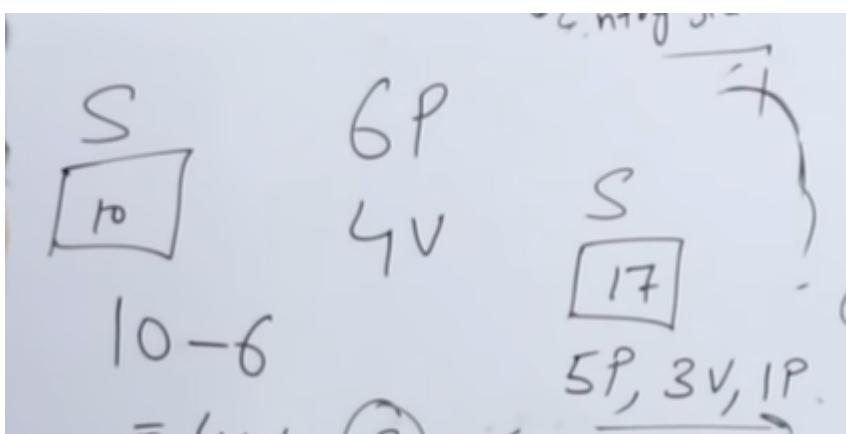
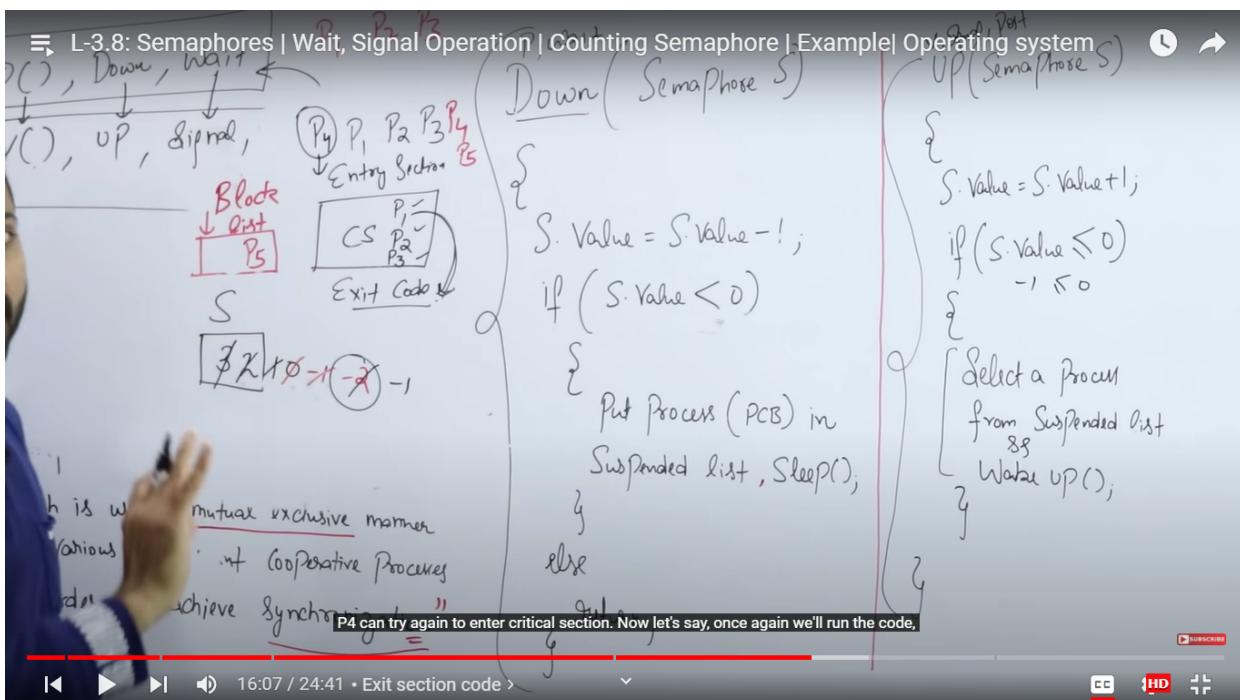


Semaphores | Wait, Signal Operation | Counting Semaphore | Example

entry : p() Down wait,

exit: v(), up, signal, post, Release

- It's an integer variable
- which is used in an mutual exclusive way by concurrent co operative maner to achieve synchronisation
- value frpm -infy to +infy and in binary from (0 ,1)
- Common code is inside the Critical section and entry code and the exit coide written OUTSIDE the CS



$$7+9=(8) \checkmark$$

$$17-5+3-1 = (14) \checkmark$$

What is Binary Semaphore

- Here s deals with only 0 and 1

L-3.9: What is Binary Semaphore | Easiest Explanation | Operating system

Binary Semaphore

Down(S)

Up(S)

if (S.Value == 1)

S = 0;

else

Block this Process And Place in Suspend List, Sleep();

Select a Process from Suspend List and Wake up();

it will put it in the ready queue, so that its turn comes.

11:38 / 12:22 • Up Operation

L-3.10: Practice Question on Binary Semaphore in Operating System

execute the following code

repeat

P(mutex)

CS

V(mutex)

forever

Exit Section

try Section

repeat

V(mutex) ← Entry

CS

V(mutex) ← Exit

forever

Carry II

P₁ P₁₀ P₂ P₃ P₁₀

12:30 / 15:59 • Case 2

L-3.10: Practice Question on Binary Semaphore in Operating System

Each process P_i ($i=1$ to 9) executes P_{10}

Processes P_{10} execute

19/27

execute the following code | the following code

```

repeat
    P(mutex)
    CS
    V(mutex)
forever
    
```

```

repeat
    V(mutex) ← Entry
    CS
    P(mutex) ← Exit
forever
    
```

Ques. What is maximum number of processes that may present in CS at any point of time?

Solution of producer Consumer problem:

L-3.11: Solution of Producer Consumer Problem using Semaphore | Operating System

Counting Semaphore full = 0

Binary Semaphore S=1;

Producer_item(item b);

down(Empty);

down(S);

Buffer[IN] = item b;

IN = (IN+1) mod n;

UP(S);

UP(full).

as well as the counting semaphore. Here I have taken 2 counting semaphore, full and empty.

N=8

0	a
1	b
2	c
3	
4	
5	
6	
7	

Consumer =

down(full);

down(S);

item C = Buffer[OUT];

OUT = (OUT+1) mod n;

UP(S)

UP(Empty);

$$\begin{aligned}
 \text{Empty} &= 545 \\
 \text{full} &= 343 \\
 S &= +\phi+\phi 1 \\
 IN &= (3+1) \bmod 8 = 4 \bmod 8 \\
 &= 4
 \end{aligned}$$

Reader and Writer problem using the semaphores:

In this problem reader and write can not enter the database or it will create the problem with the data! to avoid this process is synchronised with the semaphores as below:

Reader-Writer Problem

Diagram: A box labeled "DB R, Same Data." with two arrows pointing down from it, labeled R_1, w_1 and R_2, w_2 . To the left, there are two circles labeled "A" and "B" with arrows pointing up to the DB box.

Notes:

- Semaphore mutex = 1;
- Semaphore db = 1;
- Void Reader(Void)
 - { while (true)
 - down(mutex);
 - rc = rc + 1;
 - if (rc == 1) then down(db);
 - up(mutex);
 - }
- Process_data
- Void Writer(Void)
 - { while (true)
 - down(db);
 - DB
 - up(db);
 - }

Bottom of the video frame shows a progress bar at 4:05 / 21:10 and the title "Introduction".

Dining Philosophers problem:

5 : Philosophers

5 : Fork

Diagram: Five philosophers (P_0, P_1, P_2, P_3, P_4) are seated around a circular table with five forks (F_0, F_1, F_2, F_3, F_4). In the center is a "Rice Bowl".

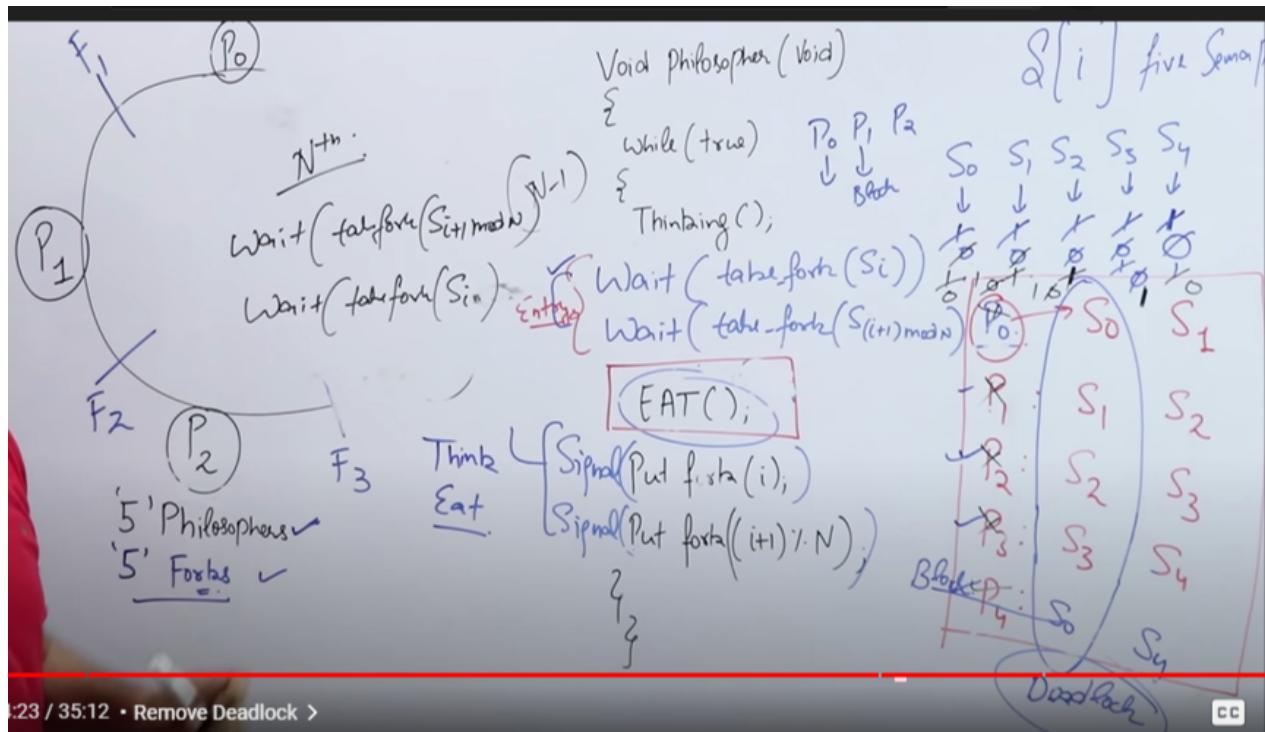
Notes:

- Void philosopher (void)
 - { while (true)
 - Thinking();
 - Wait(take_fork(S_i));
 - Enter
 - Wait(take_fork($S_{(i+1) \text{ mod } 5}$));
 - EAT();
 - Signal(Put_fork(i),);
 - Signal(Put_fork($(i+1) \cdot N$),);
- $S[i]$ five Semaphores
- State transitions for semaphores S_0, S_1, S_2, S_3, S_4 :
 - $S_0 \rightarrow S_1, S_1 \rightarrow S_2, S_2 \rightarrow S_3, S_3 \rightarrow S_4, S_4 \rightarrow S_0$
 - $S_0 \downarrow, S_1 \downarrow, S_2 \downarrow, S_3 \downarrow, S_4 \downarrow$
 - $S_0 \uparrow, S_1 \uparrow, S_2 \uparrow, S_3 \uparrow, S_4 \uparrow$

Even with the semaphore also it might get blocked and deadlock will occur if in case process gets preempt after first semaphore condition

so to avoid the deadlock we will reverse the condition for the last philosopher in which he will go to pick up the right fork first instead of left and it will get blocked and then P_3 can eat() with fork 4 and then cycles continues and finally P_4 will come out and will eat!

Note : We can apply above condition to any one philosopher!

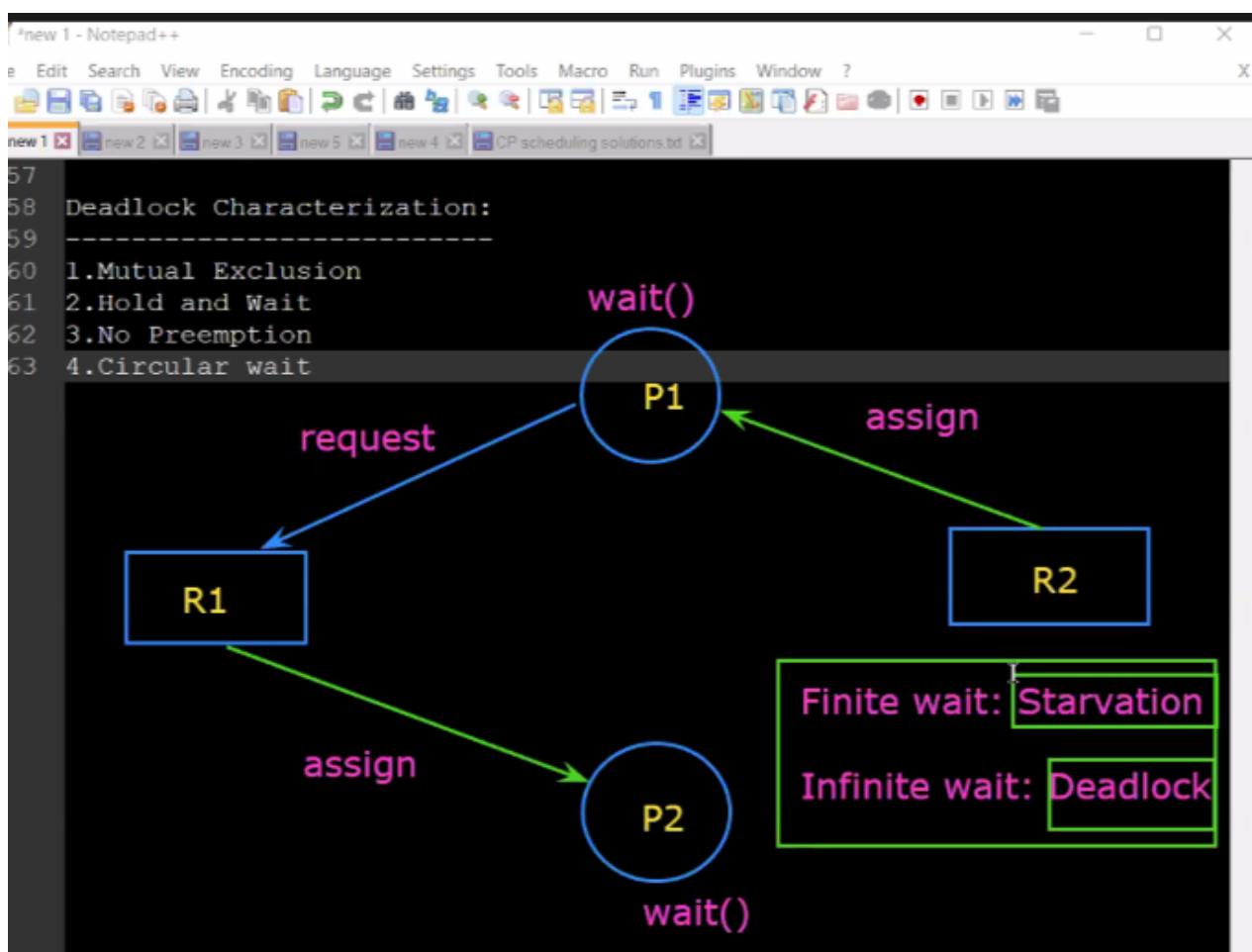


Deadlock :

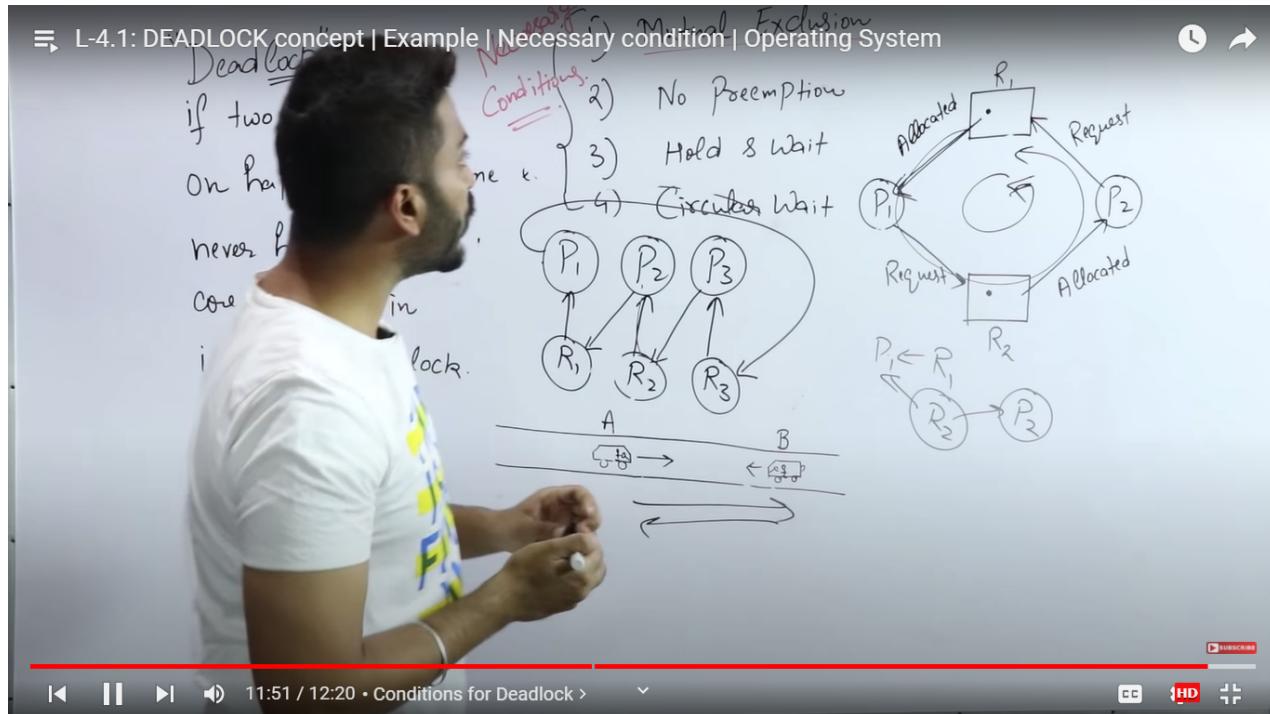
If two or more process are on happening some event which never happens :

A process in OS uses different resources and uses resources in following way:

- Request a R
 - Use R
 - Release R



Four necessary condition for deadlock:

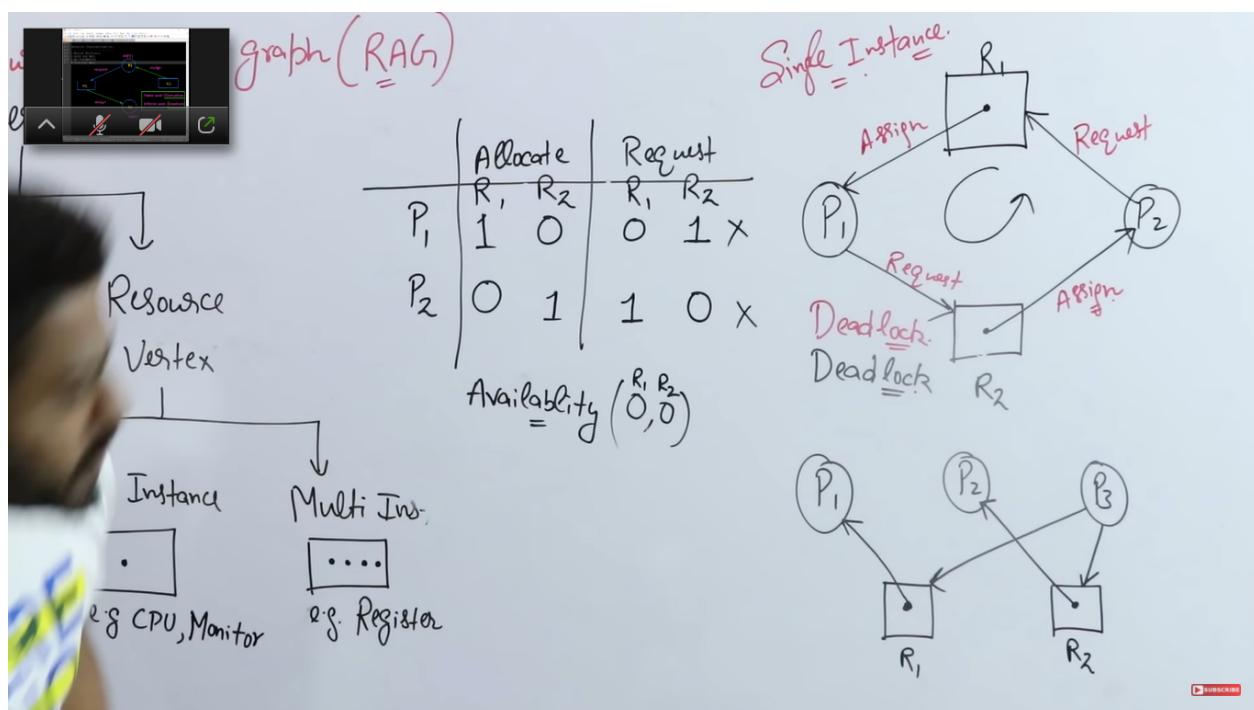


RAG : Resource allocation graph:

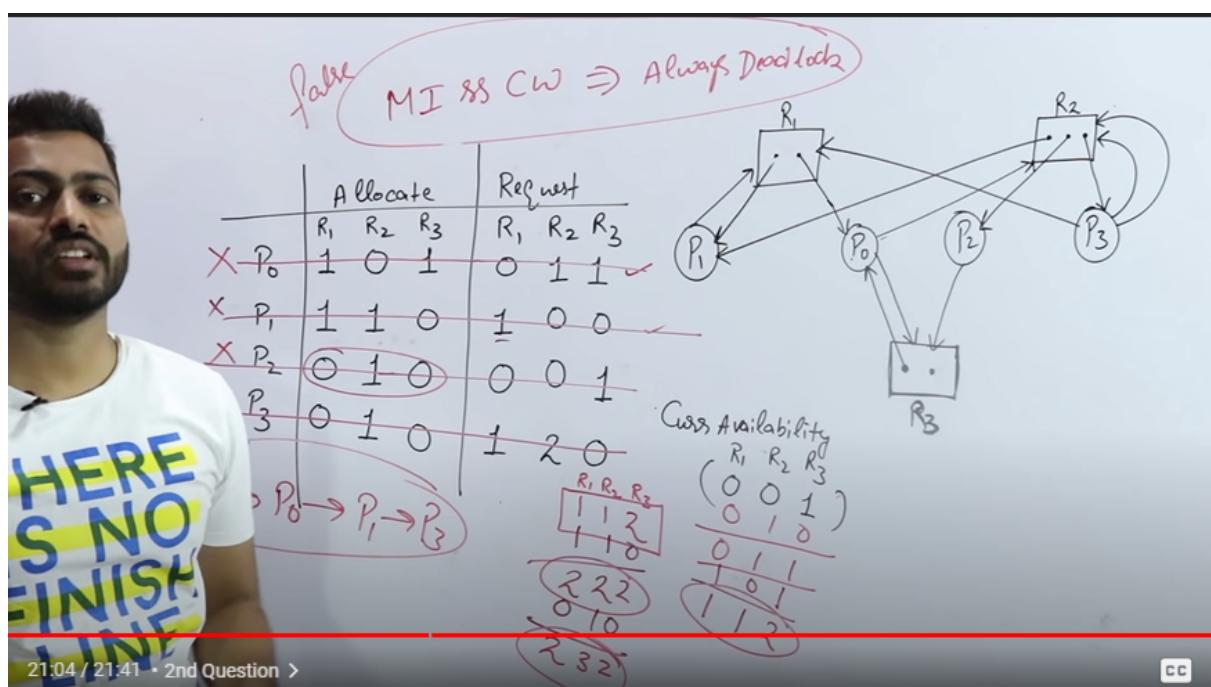
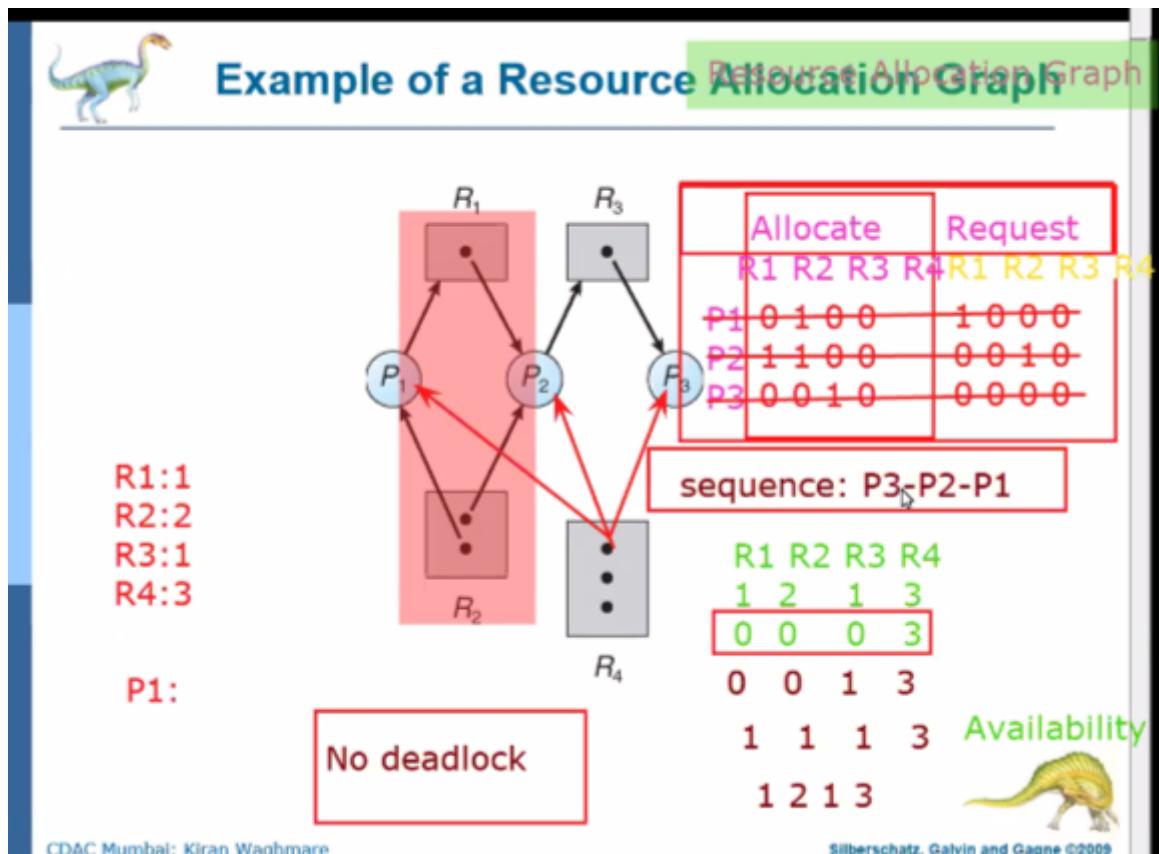
Most efficient and convenient way to represent the state of the system.

representation of how the resources have been allocated to the processes and how the processes has been assigned the multiple processes!

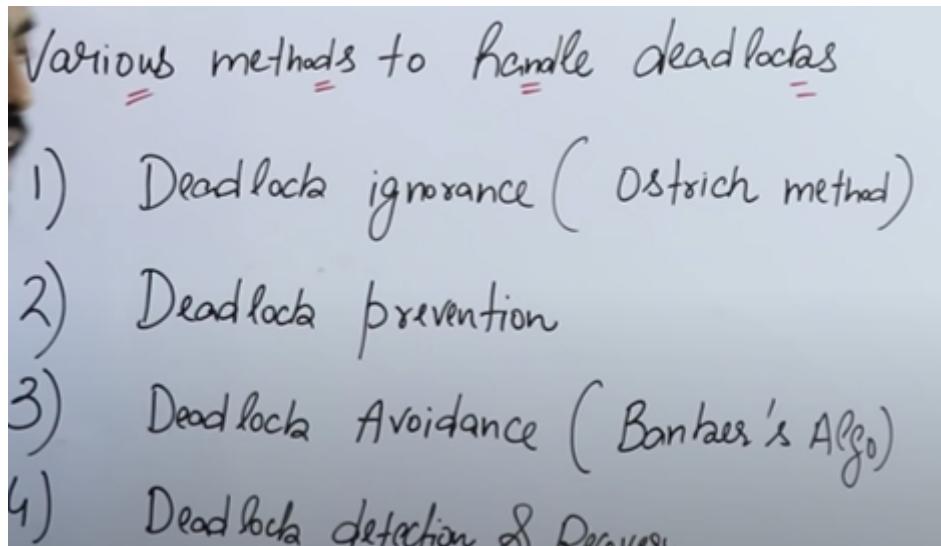
RAG is used to check if we have deadlock in the system or not



In single instance verted for cyclic wait there will be always deadlock.



Various methods to deal with the deadlock:



1. Deadlock occurs very rarely i.e. once in 5-6 years so developers have decided to ignore the deadlock instead of writing the heavy code for it and to avoid performance damage it's better to ignore it e.g. sometimes the deadlock is resolved only when we restart our PC
2. DP : In this case any process can access the resources in increasing order only with this circular wait situation is avoided
3. Banker's algo resolves the deadlock issue
4. Deadlock detection and recovery : It's a bit complex system in which

Deadlock avoidance with the banker's algorithm:

L-4.5: Deadlock Avoidance Banker's Algorithm with Example | With English Subtitles

BANKER'S ALGO

Process	CPU Allocation			Memory Allocation			Available	Remaining Need	Safe Sequence
	A	B	C	A	B	C			
P ₁	0	1	0	7	5	3	3	3	7 4 3 P ₁
P ₂	2	0	0	3	2	2	5	3	1 2 2 P ₂
P ₃	3	0	2	9	0	2			6 0 0 P ₃
P ₄	2	1	1	4	2	2			2 1 1 P ₄
P ₅	0	0	2	5	3	3			5 3 1 P ₅
	<u>7 2 5</u>								

This is not possible in real life as in reality processes demand for the resources in dynamic manner so it becomes difficult to manage the things!

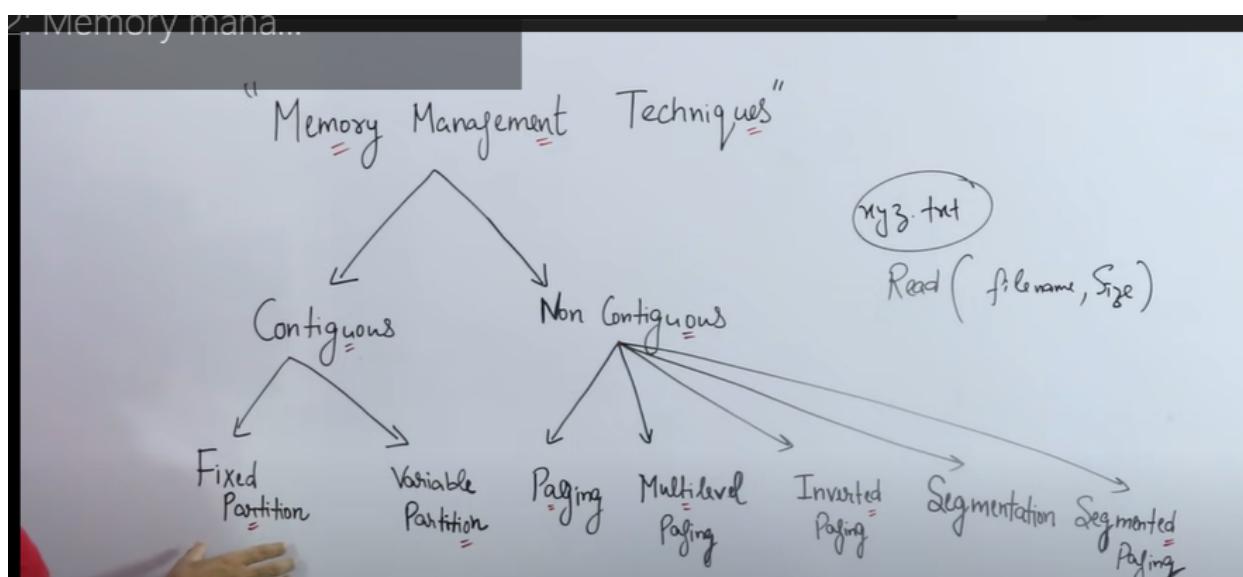
Safe state / sequence : In this case all processes get the required resources and all process gets terminated one after the other, in this case there is deadlock.

*NATE 2018
Mark*

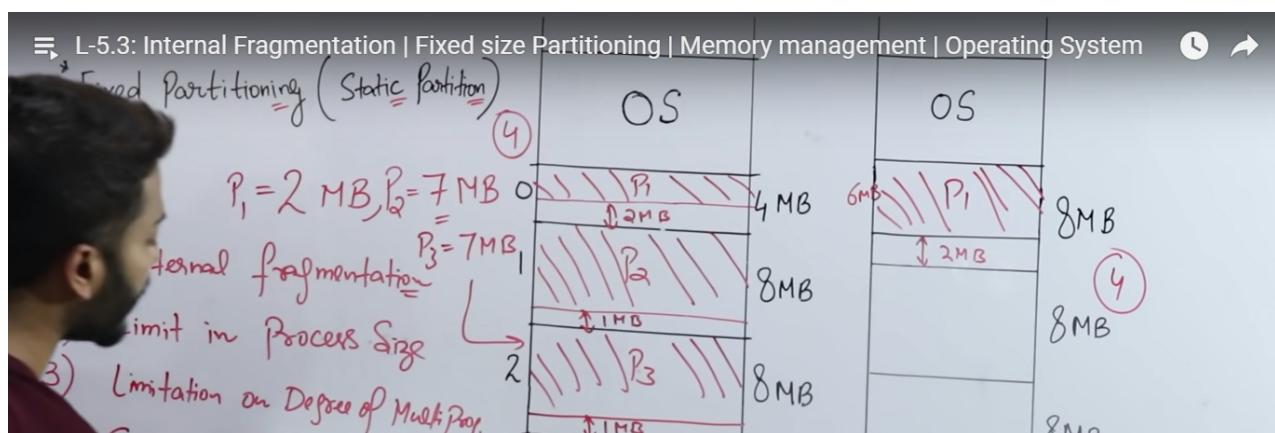
= Consider a system with 3 processes that share 4 instances of same resource type. Each process can request a max of 'K' instances. The largest value of 'K' that will always avoid deadlock is 2

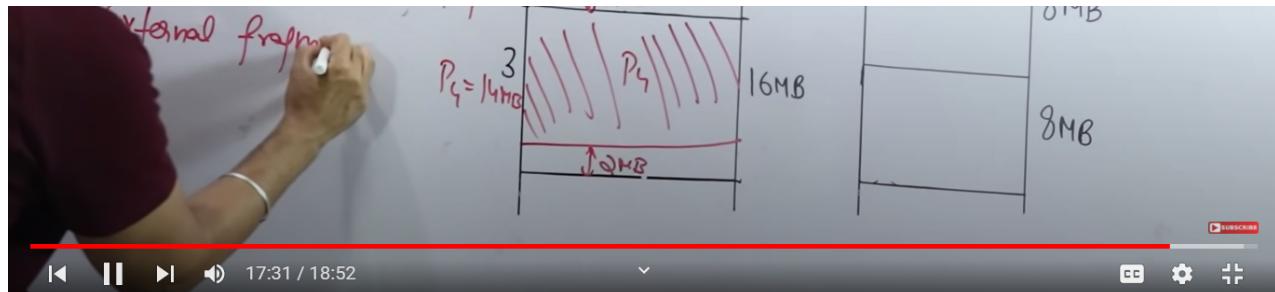
Memory Management and Degree of Multiprogramming

Degree of multiprogramming



Internal Fragmentation | Fixed size Partitioning





Variable size Partitioning

