Date: 23/09/2021

Abhishek Sharma Roll

16. Multi Level Queue Scheduling and Multilevel Queue Feedback.

We have read many scheduling Algo's like. (Every one has its own Advantates).

i) Round Robin Scheduling Algo: Good in terms of Response time.

ii) Shortedest job Scheduling Algo: Good in terms of waiting Time.

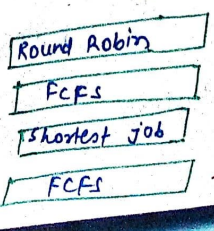
iii) FCFS Scheduling Algo: First come first serve manner.

Multilevel Queue Scheduling.

This Scheduling Algo. suit all the previous scheduling Algos. Here we devide our Ready Queue into multiple Queues. and accordig to the natures of process (Ex. certain process they need good Response Time, we put them in one Queue and Apply Round Robin scheduling there, if certain processes then have to be done in FCFS manner then we put them into one Queue and apply FCFS Shedulig there).

Here in Multilevel Queue, we have different Queues, so we have to Allocate these Queues in CPU, there might be different ways to Allocate these Queues to CPU.

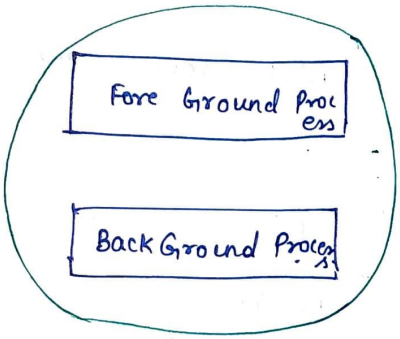
One simple ~~Queues~~ Method is to put allocate these Queues based on priority. The process which have highest priority



Abhishek Sharma Notes 6290903490

So CPU will finishes Highest priority job in the Highest Priority Queue and once it doesn't have anything to do in Highest Priority it goes to Lowest Priority and if it finishes it gone to next Queues and so on.

Another Implementation of Multilevel Queue



⇒ Fore Ground Process have Higher Priority and Background Process has the Lower Priority

In System Like UNIX This Type of Implementation is done.

* Multilevel Queue with FeedBack.

This is the most used scheduling Algorithms. Like in windows, macOS, ~~Kinux~~, They all use this scheduling Algo.

Q) Why we use Multilevel Queue with Feeback scheduling Algo?

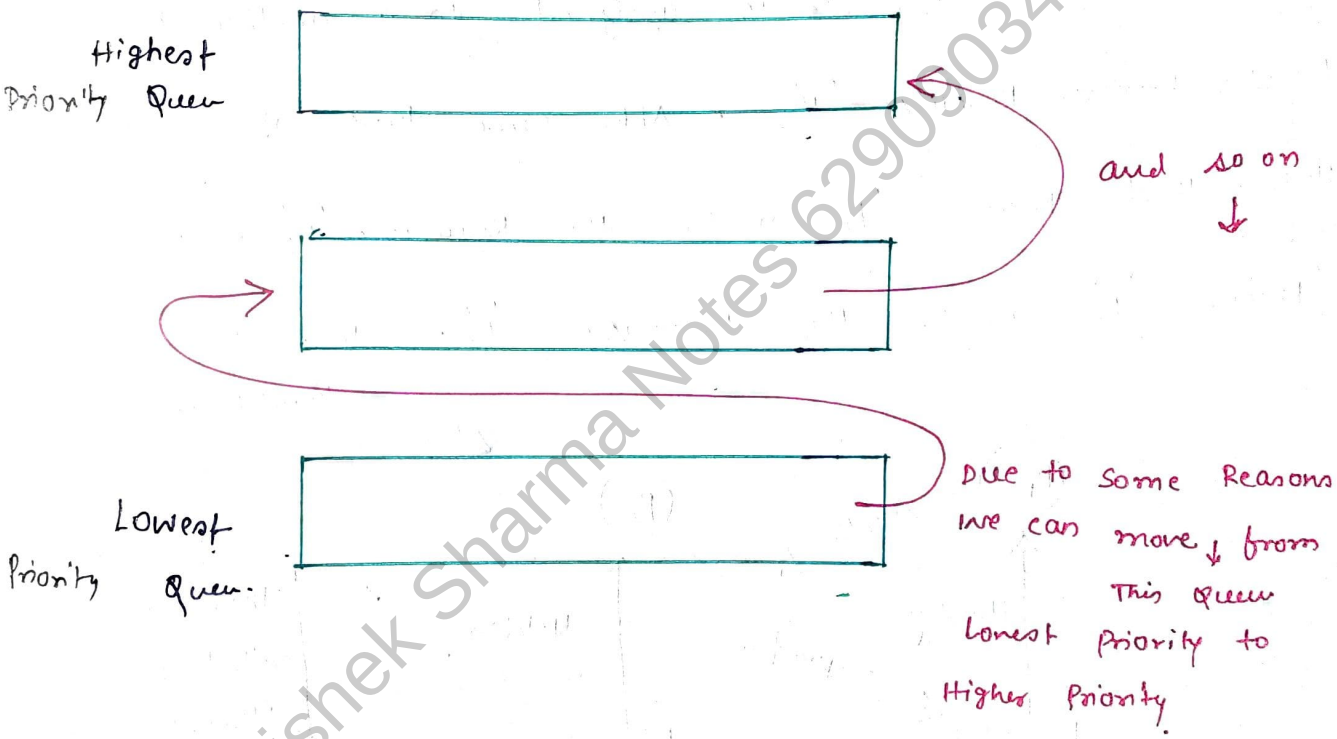
A) Bcz in Multilevel Queue scheduling starvation takes place. (Ex. If a process 'A' is in Higher Priority and a Process 'B' is in Lowest Priority. But a certain Time Process B's Priority has been changed and it is in Highest Priority. So we have to change it's Priority and gives the Processer to work upon B as it is Highest

Abhishek Sharma Notes 6290903490

Idea ↓

So If a Process is waiting in the Lowest priority Queeue and we need to Change its priority fom Lowest to Highest then we use. Multiple Queeue with FeedBack.

Idea: Due to some Reasons we can move processes from Lowest Priority to Highest Priority.



This Multilevel Queeue with FeedBack is most complex to implement bcz. we need to take decesions when the Processes to move along its Priority and Thats why it is more complex to implement. You have to take many decisions. and This is the most used Scheduling Algo.

Abhishek Sharma Notes 6290903490

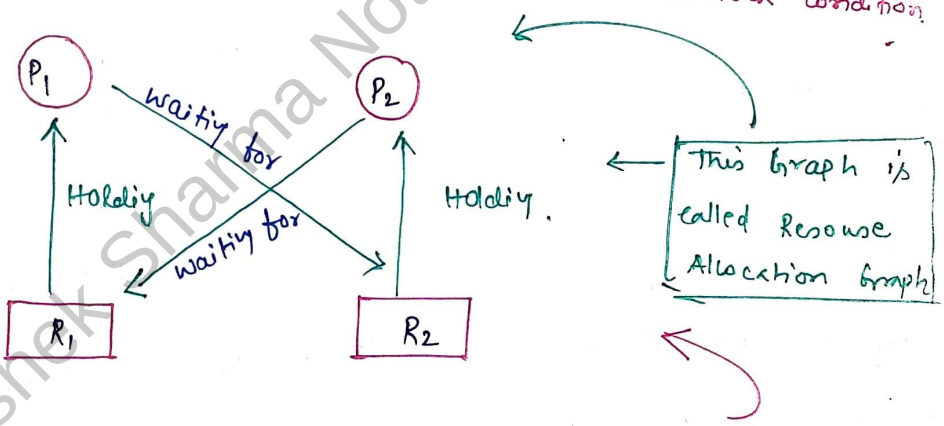
17. Dead Lock.

Abhishek Sharma Notes.

In our system there are Non-sharable Resources are available. (Like. Printers, non sharable Files) and etc.

Concept of Deadlock.

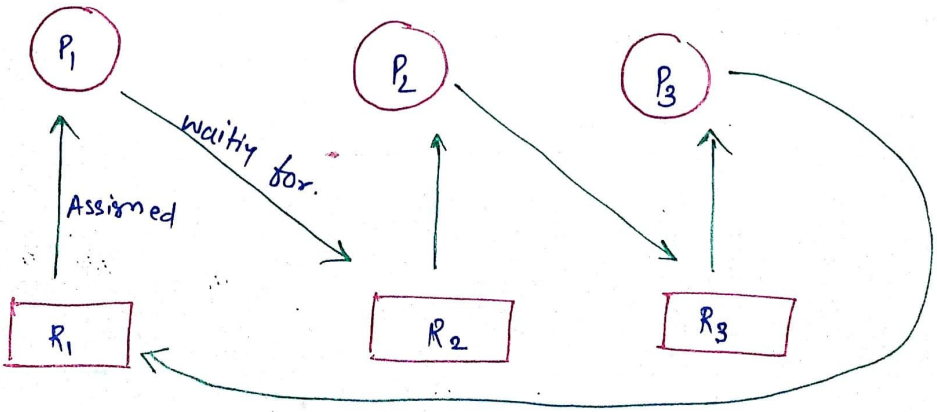
If our system assign a process 'P_1' to a non sharable Resorces. 'R_1'. and an another process 'P_2' is assigned to a another non Sharable ~~Process~~. Resourse 'R_2'. After some time it might Happen that process 'P_1' needs the Resorurse 'R_2' and 'Process 'P_2' needs the Resourse 'R_1'. This is called Dead lock condition.



Deadlock Condition with 2 Process.

Example of Deadlock with 3 processes.

Resouse Allocation Graph.



Abhishek Sharma Notes 6290903490

Here in this 3 processes Example. → Process 'P_1' is assigned to R_1 and P_2 to R_2 and P_3 to R_3. After some time P_1 need R_2 and P_2 need R_3 and P_3 need R_1.

They all are waiting in circular manner. Simultaneously.

* Conditions for Deadlock. (4 necessary conditions).

1) Mutual Exclusion. (Resources are non sharable).
2) Hold and wait (Processes must be holding some Resources and must be waiting for Resources)
3) NO Resource Preemption (OS, assigned a resource to a process and it can't be taking back).
4) Circular wait. (Processes are waiting in circle for each other by OS)

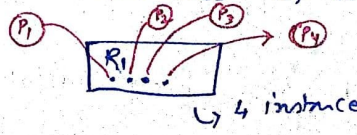
All these conditions are necessary for Deadlock be happen.

If any conditions fails then Deadlock will not happen.

Real world Example for Deadlock.

Consider a Railway Track, where 2 trains are moving from each others opposite side. at a time they will be in front of each other and can't move forward. This is the example of deadlock.

In Resource Allocation Graph we show Resources in Rectangle (□) and we show processes as circle (○) and we show the edge via (⟶). There might be more instances in a Resource we will show them using (.) Dot



↳ 4 instances.

* Deadlock happens in DBMS Algo.

Abhishek Sharma Notes 6290903490

18. Methods of Handling Deadlock

Abhishek Sharma Notes

* Deadlock Handling Methods.

i) Deadlock Prevention. :-

(In this Method we prevent one of those 4 necessary conditions. If we prevent any of those conditions then Deadlock will not happen.) → For this we have 4 methods (we will talk later.)

ii) Deadlock Avoidance :-

(When a Request comes to Operating System then OS checks this Request whether this Request causes a Deadlock or not and then it carefully grants those Request.) → for this we have an Algorithm called Banker's Algorithm (we will talk later).

iii) Deadlock Detection and Recovery :-

(If Deadlock Happens then it fixes a process and Delete it so that Deadlock will not Happen again)

iv) Ignore the deadlock :

most used.

(Simply Ignore the Deadlock if it takes lots of effort to free the process from Deadlock).

The ignorance Approach is used by unix and windows System both. They ignore the Deadlock Problem all ~~Eg~~ Together

Ex. `lockf` : This system call for unix. After this system call OS check whether granting a request will lead to Deadlock or not. if it leads to deadlock then it will show `EDEADLK` error massage.

Abhishek Sharma Notes 6290903490

19. Deadlock Prevention.

Abhishek Sharma Notes

Prevent / Eliminate any of the Following Four:

i) Mutual Exclusion: We can't eliminate this Completely. (BCz we can't eliminate a Printer to print anything as it is a non Sharable Resource). We can minimize via spooling. Spooling is a process where every job is sed to a Ready Queue and is waiting for the Resource.

ii) Hold and wait: We can ~~Hold~~ eliminate Hold and wait.

There are 2 ways to eliminate this Process:

One way: A process has to tell in Advance what all recources its going to need. (it is impractical Approach for a Process to declare all the Resources it will use in future). ← 1st Method to eliminate Hold and wait Process.

2nd way: A Process if it is requesting for a Resource. It must release all the Resource it is Holding Previously. So that new Requst can be done. ~~this~~ method is also Wastage of Approaces as well. ← 2nd Meth. for eliminate Hold and wait Process.

iii) NO Preemtion: Your OS can take away the resources that is allocated to a process. This is also problematic.

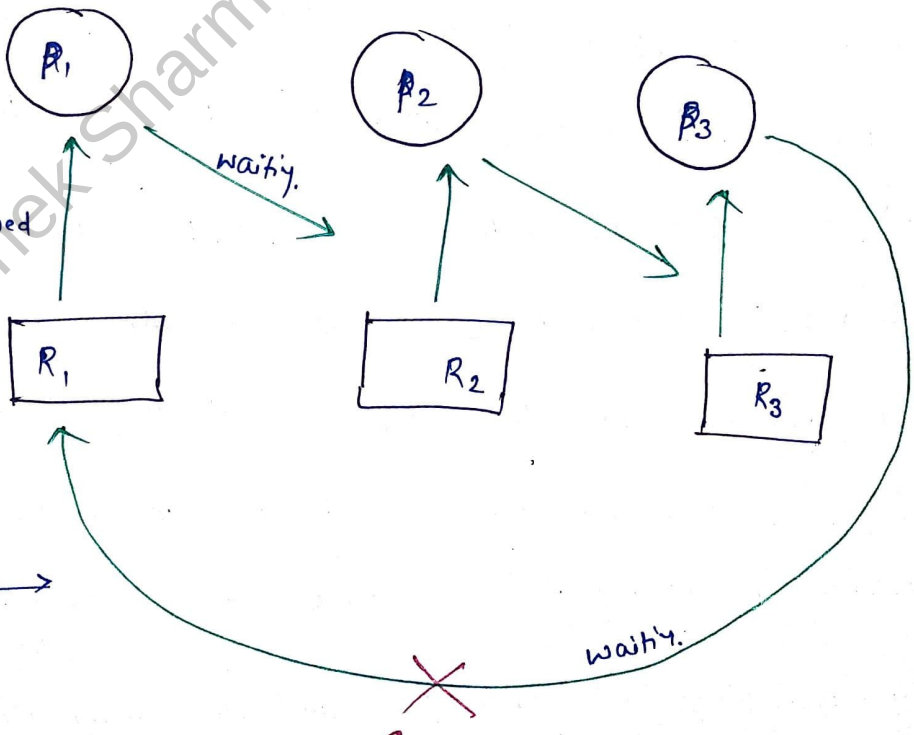
← This also looks impractical solution. bCz. it might Happen that Recourse might be in middle of a Process and we can't take away it from these.

Abhishek Sharma Notes 6290903490

iv) **circular wait:** We eliminate circular wait in our system.

Q) How can we eliminate ? Ans:. We number the Resources. We say that it is Resource no. 1, Resource no. 2, and Resource no. 3. and then we make a rule that every Process ~~is~~ can take the Resources only in increasing Order of their Number. A Process can't take Resource 1 if it is already taken Resource 2. Simultaneously if a Process take Resource 2 if it ~~is taken~~ can't take Resource 3.

Example:



This Edge is not allowed.

~~It~~ This can't happen. bcz. if a process is Holding a Higher no. of resource (R_3) Then it ~~will~~ can't request for a Lower no. of Resource (R_1).

Abhishek Sharma Notes 6290903490

20. Deadlock Avoidance - Banker's Algorithms.

Deadlock Avoidance means we learnt in our previous video. that whenever a request comes to the OS. it checks this request whether this Request causes a deadlock or not and then it carefully grants those request.

This is done by Banker's Algorithms.

Let's discuss Banker's Algorithms.

	Alloted		Max.	
	R_0	R_1	R_0	R_1
P_0	0	1	7	5
P_1	2	0	3	2
P_2	3	0	9	0
P_3	1	1	2	2
P_4	0	0	4	3

Total = < 10 , 5 >
R_0 R_1

Q consider the senerio and a request from P_3 for <1, 0> should OS grant this request ??

Sol: 1st check.

We will first check P_3 <1, 0> is not greater than its Max.

P_3 <1, 1> Previously.

P_3 <1, 0> Newly.

$\Rightarrow$ Total P_3 < 2, 1 > which is less than P_3 <2, 2> max value

So we can insert P_3 and OS grants this request.

Abhishek Sharma Notes 6290903490

2nd check.

Is the P_3 requesting value $<1, 0>$ is less than the Available value or not ??

Total = $<10, 5>$ (Given)
R_0 R_1

Available = $<10-6, 5-2>$

= $<4, 3>$

And P_3 = $<1, 0>$

So, P_3 is less than Available value.

So we OS can grant this request.

3rd check ?

Main Part of this Algorithm

Your OS Assumes that the resources (P_3 $<1,0>$) are allocated then it check what will happen after allocate. Do I have safe state after Allocation or not ??

If it is in safe state then OS will grant this request otherwise it will not granted by OS.

Q. How to check whether it is in safe state or not?

Ans If the system generates a safe sequence. then it is in safe state otherwise not in safe state

Abhishek Sharma Notes 6290903490

Safe Sequence: Permutation of all the process.

Q How to find it creates a deadlock or not. ??
Or it is in safe state, safe sequence is in or not ??

Algorithm

first Find **need** table.

need = Max − Allocated.

= <7, 5> − <0, 1>

= <7, 4>

Our System

Deadlock is a subset of unsafe state.

So From the table previously our need table will be look like.

	Allocated.		Max.		need	
	R_0	R_1	R_0	R_1	R_0	R_1
P_0	0	1	7	5	7	4
P_1	2	0	3	2	1	2
P_2	3	0	9	0	6	0
P_3	2	1	2	2	0	1
P_4	0	0	4	3	4	3

Algorithm says that

Safe_seq. = { }

While (All P_is are not added to the safe_seq.)

{

a) Find a P_i such that

need $_i$ ≤ available

b) If (no such i exists)

return false;

Continued.

Abhishek Sharma Notes 6290095900

```
else (we found an i)
{
    available += allocate i  Add Pi to the safe_seq.;
}
}
return true;
```

Total = <10, 5>

Initial available <3, 3> (BCZ P_3 request <1,0> include)

Now we will go through the procedure and our table process one by one

In starting safe_seque. = { } (containing null value)

Let's start with P_0. its need is <7, 4> and available is <3, 3> so it's need is not ≤ available.

Now,

P_1 its need is <1, 2> and Available is <3, 3>

it satisfies the criteria. need$_i$ ≤ available

Now we will update available as available of P_1 + allocate of P_1

<3+2, 3+0> = <5, 3>

Our update available = <5, 3>

now P_1 is added to safe_seq. = { P_1 }

Now,

P_2 it's need is <6, 0> But available is <5, 3>

So iterate the loop.

Abhishek Sharma Notes 6299903490

Now, P_3 its need $<0, 1>$ and available is $<5, 3>$. So it satisfies our criteria

So we add ~~available~~ Allocated. of P_3 to our current available.

$<5+2, 3+1> = <7, 4>$

Now our current available $= <7, 4>$

and P_3 is added to our safe_seq = $\{P_1\ P_3\}$

Now, we will iterate and P_4 is checked.

its need $= <4, 3>$ and available is $<7, 4>$

so it matches our criteria. Adding Allocated of P_4 to our current Available.

so $<7+0, 4+0> = <7, 4>$

Now our current available $= <7, 4>$

Now, P_4 is added to our safe_seq = $\{P_1\ P_3\ P_4\}$

Now again the loop iterates. (it will iteratate till all the Process are not added in the safe_seq.).

Now we will check for P_0.

its need $= <7, 4>$ and our current available $= <7, 4>$

Condition matched. Now P_0's Allocated will be addded to current Available

so, $<7+0, 4+1> = <7, 5>$

Now our current available $= <7, 5>$

P_0 is added to our safe_seq = $\{P_1\ P_3\ P_4\ P_0\}$

Now check for P_2 its need $= <6, 0>$ and current available $<7, 5>$

current availbe $= <10, 5>$ P_2 is also added = $\{P_1 P_3 P_4 P_0 P_2\}$
safe_seq=

So Now All the Process are in safe seq. So request has been granted.

Abhishek Sharma Notes 6290903490