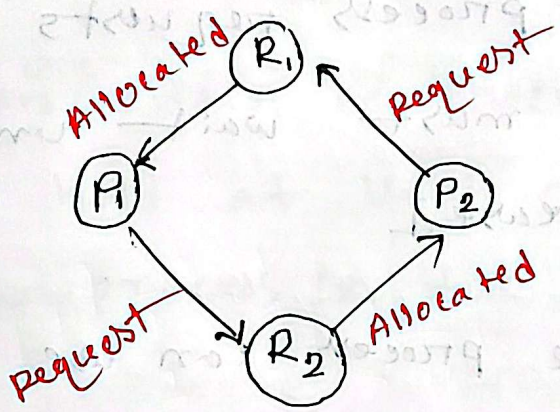
Dead lock

Dead lock occurs when a process or thread enters a waiting state because a requested system resource is held by another waiting process which in turn is waiting for another resource held by another waiting process.



Condition for occurring Dead lock

(I) Mutual exclusion
(II) Hold and wait
(III) No preemption
(IV) Circular wait

Necessary condition

There four condition must ~~be~~ all hold simultaneously for a deadlock to occur.

① Mutual Exclusion:

Only one process can use a resource at a time.

If another process requests the same resource, it must wait until the resource is released

Example: Only one process can use a printer at a time.

② Hold & wait:

A process is holding at least one resource and waiting to acquire additional resource held by other processes.

Example: process P_1 is holding a printer and waiting for a scanner.

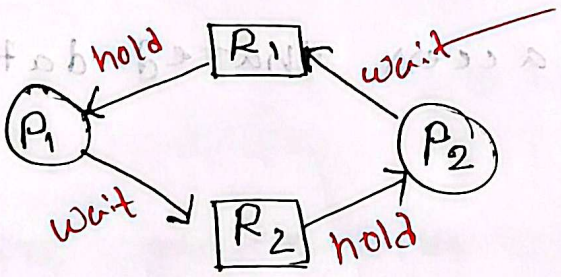
No Preemption:

(iii) Resources cannot be forcibly taken away form a process, they must be released voluntarily by the process holding them.

Ex: The OS cannot take a printer from a process until it releases it.

(iv) Circular wait:

A circular chain of process exists, where each process hold at least one resource that the next process in the chain needs.



here, P_1 → hold R_1, wait for R_2

P_2 → hold R_2, wait for R_1

→ There exits a set of waiting $(P_0, P_1 \cdots P_n)$ processes such that: P_0 is waiting for a resource that is is held by P_1, P_1 is

waiting for a resource that is held by P_2 ..., P_{n-1} is waiting for a resource that is held by P_n and P_n is waiting for a resource that is held by P_0.

Deadlock with mutex & locks

mutual exclusion locks.

A mutex lock is a synchronization mechanism used to protect share resources from being accessed by multiple threads or processes at the same time.

→ It ensure that only one thread can enter the critical section (the part of code that access shared data) at any given moment.

→ A mutex lock is like a key:—

(i) When one thread takes the key (lock) others must waits.

(ii) After finishing, it returns the key (unlock); so another thread can enter.

main purpose: To prevent ~~one~~ race conditions and ensure data consistency in concurrent (multithread) program.

ex: If two thread want to update the same file.

Thread 1 locks the file → work → unlocks

Thread 2 ~~locks the file~~ wait until thread one unlocks.

Thus, the file remain consistent and safe.

code:

```
Initialize mutex-locks

process Pi:
  while (True) do
    Acquire (mutex-lock) // enter critical section
      critical section //access shared resource
    Release (mutex-locks) //exit critical section
end while Remainder section: //perform other operation
```

(Book) code:

```
initialize the mutex;
Pthread_mutex_t first_mutex;
Pthread_mutex_t second_mutex;
Pthread_mutex_init (&first_mutex);
Pthread_mutex_init (&second_mutex);
```

→ Thread-One run in this function

```
void *do_one (void *param) {
    Pthread_mutex_lock (&first_mutex);
    Pthread_mutex_lock (&second_mutex);
    // ... Do some work
    Pthread_mutex_unlock (&second_mutex);
    Pthread_mutex_unlock (&first_mutex);
    Pthread_exit (0);
}
```

→ Thread two run in this function.

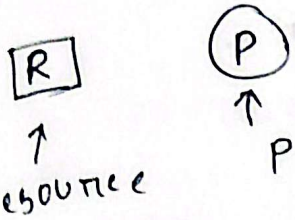
```
void *do_two (void *param) {
    Pthread_mutex_lock (&second_mutex);
    Pthread_mutex_lock (&first_mutex);
    // Do some work
    Pthread_mutex_unlock (&first_mutex);
    Pthread_mutex_unlock (&second_mutex);
    Pthread_exit (0);
```

Resource Allocation Graph

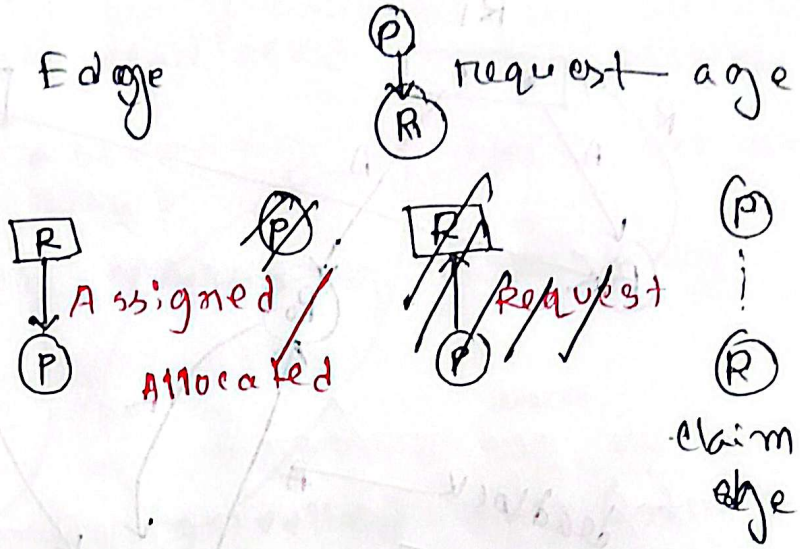
Deadlock avoidance

Graph

(i) Vertex



(ii) Edge



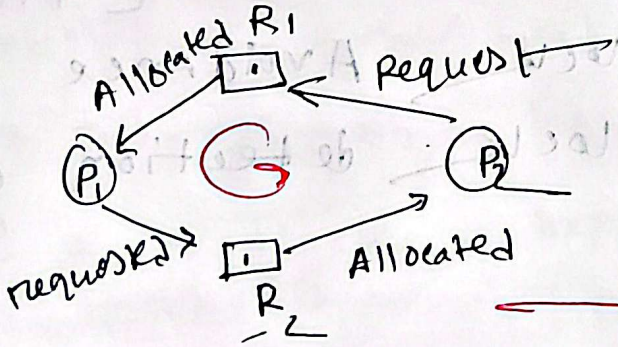
Resource:

(i) single Instance:- which has only one instance.

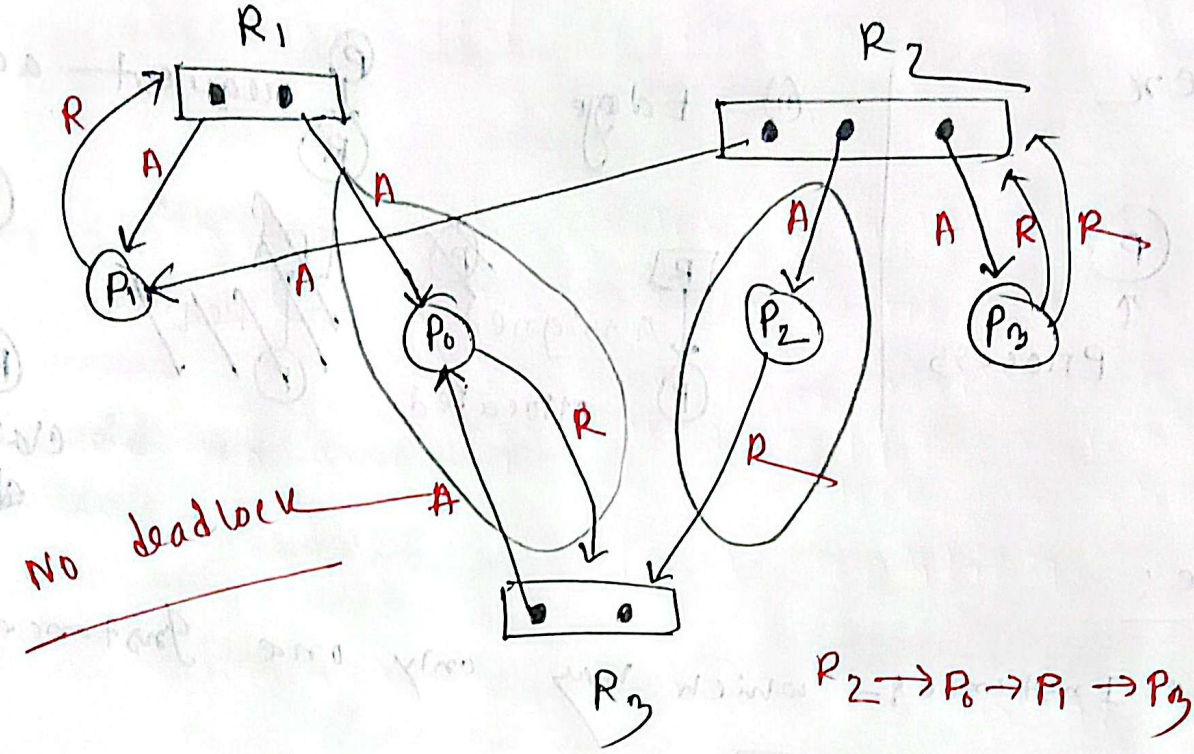
Ex:

(ii) Multi-instance: which has multiple instance.

(i) single instance:



(ii) Multi - Instance :



$P_2 \rightarrow P_0 \rightarrow P_1 \rightarrow P_3$

Dead lock Handling methods

methods

(i) Deadlock Ignorance
(ii) Deadlock ~~Preemption~~ Prevention
(iii) Deadlock Avoidance
(iv) Deadlock detection and recovery

① Deadlock Ignorance:

→ The system simply ignore the problem and pretend that deadlocks never occur in the system → assuming deadlock are rare.

→ used most OS including ~~Unix~~.

Unix and windows.

→ This approach is called the Ostrich Algorithm, Because it resembles the ostrich burying its head in the sand and pretending the problem doesn't exist.

→ If a deadlock does occur, the OS may take drastic measures such as rebooting to recover.

Advantage: No overhead or extra cost

Disadvantage: system may hang or freeze if a deadlock happens.

(11) Deadlock preemption: prevention:

→ Need to remove all condition or at least one necessary condition to prevent the deadlock

coffman condition { mutual exclusion, hold & wait, No preemption, circular wait.

→ Deadlocks have four necessary condition and they are called coffman condition. So we can prevent a deadlock by eliminating any of the above four condition.

① Eliminating mutual Exclusion:

Meaning: Only one process can use a resource at a time. where resources are non-shareable mode.

Prevention: make sure resources are shareable whenever possible.

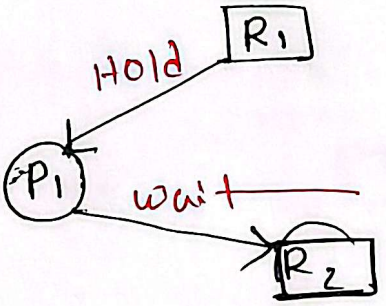
example: Read-Only files can be shared by multiple processes at the same time.

So, for non-shareable resources, prevention through this method is not possible.

② Eliminate Hold & wait:

meaning: A process is holding some resources and waiting for others.

prevention:



There are two way to eliminate hold and wait.

① By eliminating wait: The process declares all required resources before execution begins so it avoids waiting during execution.

ex: process P1 declares in advance that it requires both R1 and R2.

resources

② By eliminating Hold: The process has to release all resources it is currently holding before making a new request.

$2^{nd} = 200 + 200 = 400$ msec.

Safe state: If the system can successfully allocate to each process by giving resource then it will be called safe state.

system resource

R1 → 12

Process	required resource	Allocated resource	Resource Needed
P_1	6	2	4
P_2	11	5	6
P_3	5	3	2
	10	10	

Available Resource = Total instance/resource − Allocated resource

= 12 − 10

= 2

P_3 Q: needed resource 2

50. available resource.

P_3 process.

P_3

P_3	5	1
P_1	7	→ 1
P_2	12	1 available.

$P_3 \rightarrow P_1 \rightarrow P_2$

no deadlock occur here. that's why it called unsafe state.

Unsafe state: If the system cannot avoid deadlock by allocating resource to each process then it will be called as unsafe state.

total resource = 12

Process	Require resource	Allocated resource	resource Needed
P_1	9	3	7
P_2	11	5	6
P_3	5	3	2
		10	

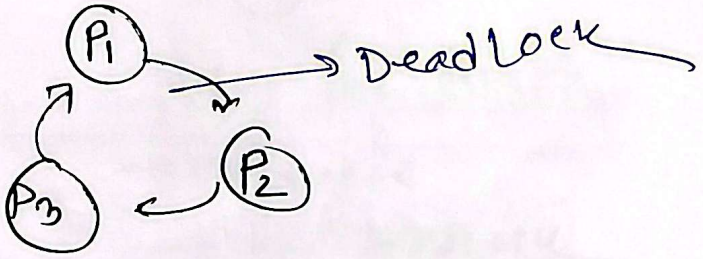
available resource = 12 - 10
= 2

P_3	5

$P_3 \rightarrow P_2$

No shareable resource. P_2 need 6 resources, but resource available = 5

thats why deadlock occurs

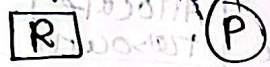


single instance → Resource Allocation Graph

Deadlock Avoidance

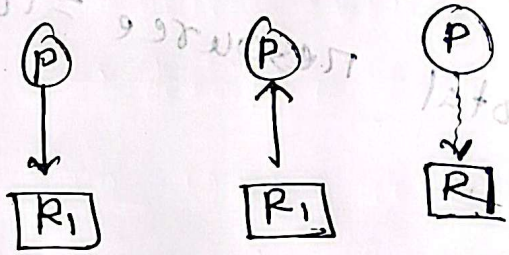
Graph:

(i) Vertex



(Resource) (process)

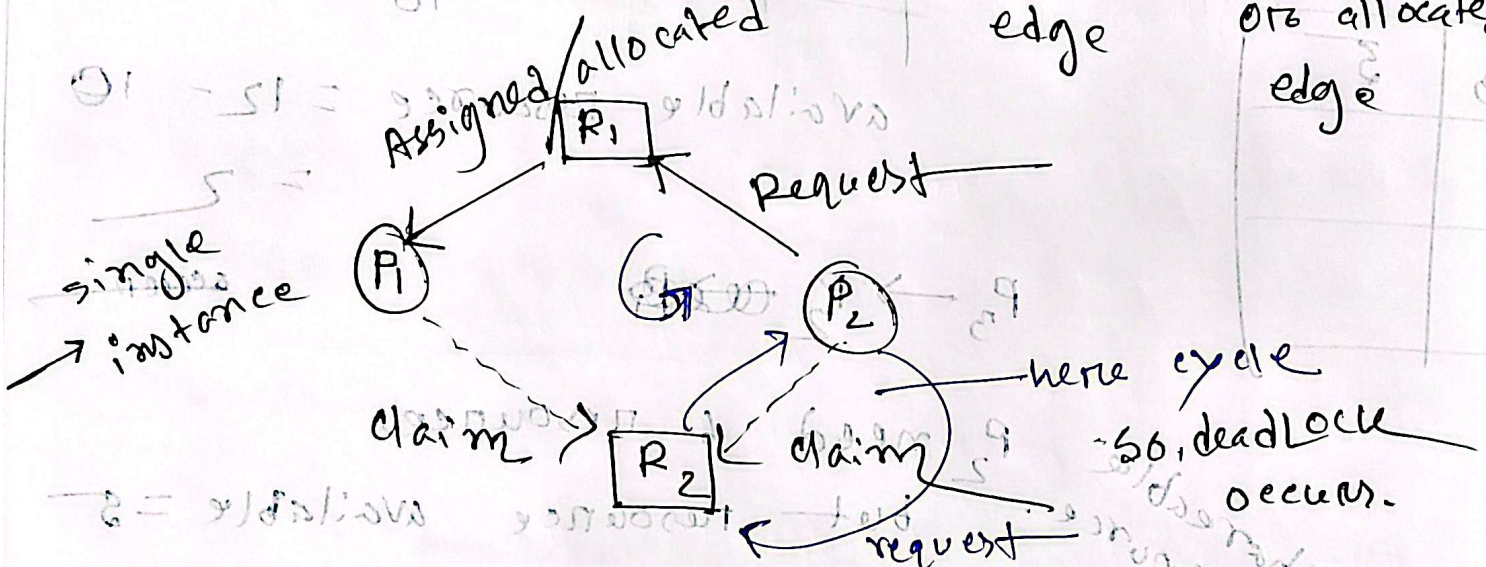
(ii) Edge

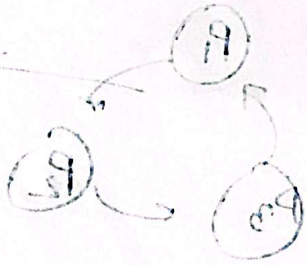


Request edge

Assigned or allocated edge

(claim edge)





Bankers Algorithm (multi-instance)

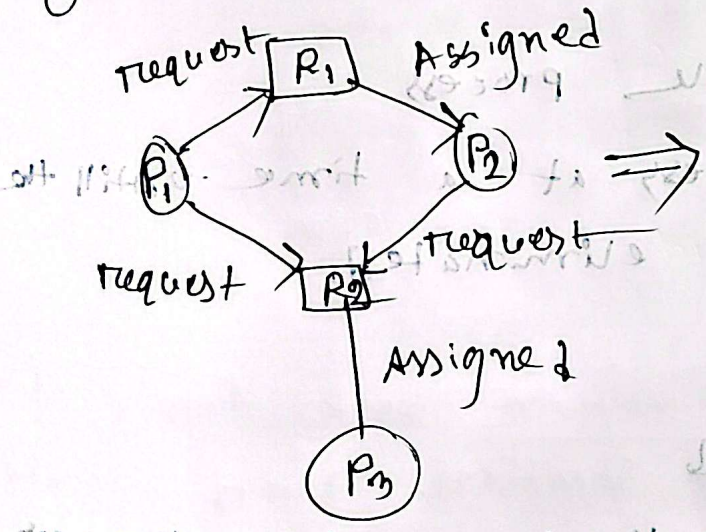
Deadlock detection & Recovery

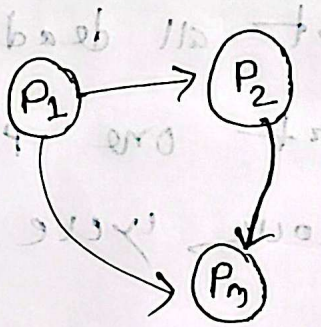
Deadlock detection

- cycle → wait for Graph (single instance)
- Bankers Algorithm (multiple instance) → working always safe state.

Example of wait for Graph:

①





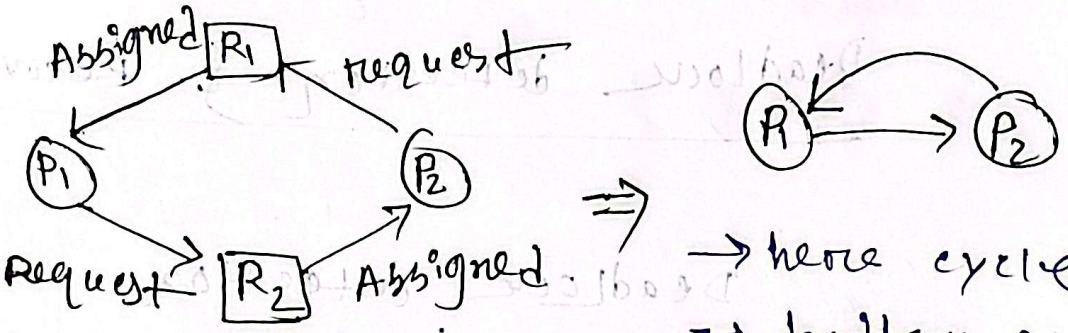
wait for Graph

→ no cycle

→ no deadlock

(Resource Allocation Graph)

(ii) wait for Graph



⇒ → here cycle
→ deadlock occur

⟶ Resource allocation Graph

তাই কোনো deadlock recovery algorithm use করতে হবে

Deadlock recovery

Process Termination:

(i) Abort all deadlock process

(ii) Abort one process at a time until the deadlock cycle is eliminated.

Resource preemption

(i) selecting a victim: — which resources & ~~process~~ process are to be preempted.

As in process termination, we must determine the order of preemption to minimize the cost.

(ii) **Rollback:** Rollback the process which was selecting as a victim

→ rollback the process to some safe state

→ abort the process and the restart it.

(iii) **Starvation:** It may happen that the same process is always picked as a victim, As a result the process never completes its task.

we must ensure that a process can be picked as a victim only a (small) finite number of time

Bankers Algorithm

$R_1 = 10,\ R_2 = 5,\ R_3 = 7$

Process	max Need			Allocated resource			current resource need			available resource		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_0	7	5	3	0	1	0	7	4	3	3	3	2
P_1	3	2	2	2	0	0	1	2	2	5	3	2
P_2	9	0	2	3	0	2	6	0	0	7	4	3
P_3	2	2	2	2	1	1	0	1	1	7	4	5
P_4	4	3	3	0	0	2	4	3	1	10	4	7
				7	2	5				10	5	7

free instances

current resource need = max need - allocated resources

Now, $10 - 7 = 3 \quad (R_1 - \Sigma R_1)$

$5 - 2 = 3 \quad (R_2 - \Sigma R_2)$

$7 - 5 = 2 \quad (R_3 - \Sigma R_3)$

safe seqn → P_1, P_3, P_1, P_2, P_0

$= 3 + 2$ (Available + allocated)

$= 5$

$2 + 0 = 2$

$2 + 0 = 2$

~~Safe process~~

Safe state: If the system successfully allocate to each process by giving resource than it will be called safe state.

resource in system → R = 12

Process	Required resource	Allocated resource	Resource needed
P_1	6	2	4
P_2	11	5	6
P_3	5	3	2
		10	

Available resource = Total instance − Allocated resource
= 12 − 10
= 2

∴ Available resource

10 টি resource, system P_1, P_2, P_3 process কে allocate করে রাখছে

Safe seq: [যে process এর resource requirement সবচেয়ে কম (যে process কে সবচেয়ে কম Available resource দিতে হয়)

$P_3 \rightarrow P_1 \rightarrow P_2$

P_3 ~~done~~	5	1 (P_3 → done execution)
P_1	7	1
P_2	12	— now avail available resource

Unsafe state: If the system cannot avoid the deadlock by allocating resource to each process then it will be called as unsafe state.

R=12

Process	Required resource	Allocated resource	Resource need
P1	9	2	7
P2	11	5	6
P3	5	3	2
		10	

available resource = 12 − 10 = 2

P3	5
P2	

P3 → deadlock

2 টি resource P3 পেলে available resource হবে 5. then P2 কে 5 টি available resource দিলে সে full execution হবে না কারণ P2 এর প্রয়োজন 6 টি but আছে 5 টি। তাই এখানে Deadlock হবে। which is unsafe.

(i)

The need matrix is calculated by

Current Need = max - Allocation

Need matrix:

Process	A	B	C	
P_0	7	4	3	
P_1	1	2	2	
P_2	6	0	0	
P_3	0	1	0	
P_4	4	3	1	

(ii) Unsafe: There is no guarantee that all ~~possible~~ possesses can finish execution in some order with the currently available and allocated resources.

Given, A = 10, B = 5, C = 7

Process	Max need			Allocation resource			current need			Available resource		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	7	5	3	0	1	0	7	4	3	3	3	2
P_1	3	2	2	2	0	0	1	2	2	5	3	2
P_2	9	0	2	3	0	2	6	0	0	7	4	3
P_3	2	2	2	2	1	1	0	1	1	7	4	5
P_4	4	3	3	0	0	2	4	3	1	10	4	7
				7	2	5				10	5	7

find available resource:

= Given instance − total allocation instance

$A_1 = A - \Sigma A = (10 - 7) = 3$

similarly $B - \Sigma B = (5 - 2) = 3$

and $C - \Sigma C = 7 - 5 = 2$

Safe seam: $P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_2 \rightarrow P_0$

∴ system is in safe state.

→ Available at start = (A, B, C) = (3, 3, 2)

(01) P_0 need (7, 4, 3) > (3, 3, 2) → cannot run now

(2) P_1 need (1, 2, 2) < (3, 3, 2) → can run

after P_1 finishes, available = (3+2, 3+0, 2+0) allocation

= 5, 3, 2

(3) P_2 need (6, 0, 0) > (5, 3, 2) → cannot

(4) P_3 need (0, 1, 1) < (5, 3, 2) → yes

P_3 finishes, ~~available~~ available = (5+2, 3+1, 2+1)

= (7, 4, 3)

similarly

new allocation = (0+0, 1+2, 0+0)

= (0, 3, 0)

$P_3 \rightarrow P_1 \rightarrow P_2 \rightarrow P_0 \rightarrow P_4$

(iii). New request from P_0 (0,2,0)

→ request = (0,2,0)

→ current need (P_0) = (7,4,3) so request(0,2,0) ≤ Need

→ Available = (3,3,2). so request (0,2,0) ≤ available

The request be granted immediately.

So, after allocating we get,

new allocation ~~(0+0, 0+2, 0)~~ (0, 1+2, 0)
= (0,3,0)

new current need (7, 4-2, 3) = (7,2,3).

new available (3, 3-2, 2-0) = (3,1,2)

process	max need			Allocated resource			current resource			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	7	5	3	0	3	0	7	2	3	3	1	2
P_1	3	2	2	2	0	0	1	2	2	5	2	3
P_2	9	0	2	3	0	2	6	0	0	7	2	3
P_3	2	2	2	2	1	1	0	1	1	10	2	5
P_4	4	3	3	0	0	2	4	3	1	10	5	5
				7	4	5				10	5	7

→ with available = (3, 1, 2)

① P_3 need (1, 2, 2) → ~~cant~~ NO

② P_3 need (0, 1, 1) → yes

available = (3+2, 1+1, 2+1)

= 5, 2, 3

~~Safe~~ Safe seen: $P_3 \rightarrow P_1 \rightarrow P_2 \rightarrow P_0 \rightarrow P_4$

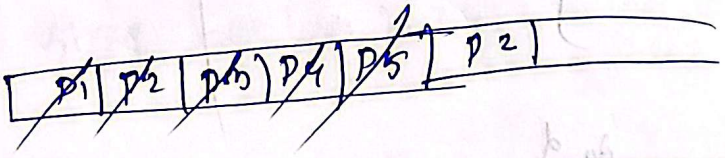
Because, such sequence exists the system is in safe state.

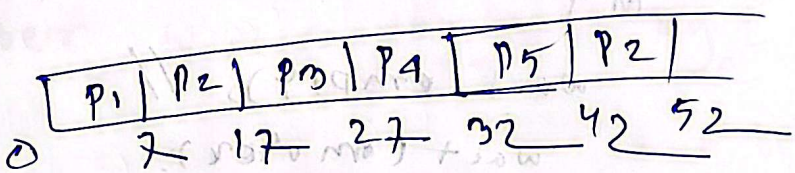
All process can be finished after the tentative allocation. So, the result (0, 2, 0) from P_0 can be granted.

Burst time

P1 ⟶ 70
P2 — 20 10 0
P3 — 10 0
P4 — 5 0
P5 — 10 0

Quantum = 10





Critical section: A critical section is the part of a process where ~~shared~~ shared resource (variable, file, device) are accessed or ~~modifi~~ modified and which must not be executed by more than one process or thread at the same time.