UNIT-3 Chapter-8

Deadlock :- In a multiprogramming environment, several processes may compete for a finite number of resources.

A process requests resources, if the resources are not available at that time, the process enters a wait state. Waiting processes may never again change state, because the resources they have requested are held by other waiting processes. This situation is called Deadlock.

System Model Under the normal mode of operation, a process may utilize a resource in only the following sequence-

① - Request - Semaphores
② - Use - Physical (printers, m/m space) or logically (files, Semaphores)
③ - Release. - Semaphores

Deadlock Characterization :-

4 Necessary Conditions -

① - **Mutual Exclusion.** - At least one resource must be held in a non-sharable mode; that is, only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.

② - **Hold & Wait :-** A process must be holding at least one resource and waiting to acquire additional resources that are currently being by other processes.

③ - **No Preemption :-** Resources cannot be preempted; that is, a resource can be released only voluntarily by the process holding it, after that process has completed ~~this~~ its task.

④ - **Circular wait. :-** A set $\{P_0, P_1 \cdots P_n\}$ of waiting processes must exist such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by $P_2 \cdots P_{n-1}$ is waiting for a resource that is held by P_n, and P_n is waiting for resource held by P_0.

Resource Allocation Graph :- It is a directed graph called RAG, where - Nodes set $P = \{P_1, P_2 \cdots P_n\}$ and $R = \{R_1, R_2 \cdots R_n\}$ set of resources. And a set of Edges E

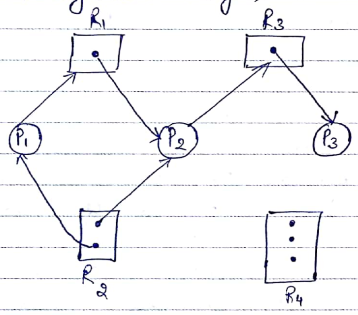
BOND WITH THE BEST

Reid & Taylor

Scanned by CamScanner

$P_i \rightarrow R_j \Rightarrow$ Process Pi requested an instance of resource type Rj and currently waiting for that resource. (Request Edge)

$R_j \rightarrow P_i \Rightarrow$ It signifies that an instance of resource type Rj has been allocated to process Pi. (Assignment Edge).



Process states –

If in a RAG, it can be shown that, if the graph contains no cycles, then no process in the system is deadlocked. If the graph does contain a cycle, then a deadlock exist.

Same graph with an edge $P_3 \rightarrow R_2$ contains a cycle and dealock will occur.

A Cycle in the graph is both a necessary and a sufficient condition for the existence of deadlock. (in case of multiple instance of resources).

Methods for handling Deadlocks – 3 ways –

① - We can use a protocol to prevent or avoid deadlocks, ensuring that the system will never enter a deadlock state.

② - We can allow the system to enter a deadlock state, detect it and recover.

③ - We can ignore the problem altogether and pretend that deadlocks never occur in the system.

Deadlock Prevention :- It is a set of methods for ensuring that atleast one of the necessary conditions can not hold.

Deadlock Avoidance :- It requires that the O.S be given in advance additional infⁿ. concerning which resources a process will request and use during its lifetime.

BOND WITH THE BEST

Reid & Taylor

Scanned by CamScanner

Deadlock Prevention :-

① - M.E :- The M.E condition must hold for non-sharable. Sharable resources, on the other-hand do not require mutualy exclusive access, and thus cannot be involved in a deadlock.

② - Hold & Wait :- We must guarntee that, whenever a process requests a resource, it does not hold any other resources.

③ - No preemption :- Preemption would be done on waiting processes.

④ - Circular wait :- Let $R = \{R_1, R_2 \cdots R_m\}$ be the set of resource types. We assign to each resource type a unique integer number;

for ex -

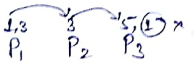
$F(\text{tape drive}) = 1$

$F(\text{disk drive}) = 5$

$F(\text{Printer}) = 12.$

" Each process can request resources only in an Increasing order of enumeration."

i.e. A process can initially request any no. of instances of a resource type, say R_i. After that, the process can request any ~~no of~~ instances of resource type R_j. if and only if $F(R_j) > F(R_i)$. If several instances of the same resource type are needed, a single request for all of them is must be issued.



Alternatively, we can require that, whenever a process requests an instance of resource type R_j, it has released any resources R_i such that $F(R_i) \geq F(R_j)$

Deadlock Avoidance :-

Prevention depends on how requests can be made. Possible sideeffects of preventing deadlocks by this method, are low device utilization and reduced system throughput.

"Each request requires that the system consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process, to decide whether the current request can be satisfied or must wait to avoid a possible future deadlock."

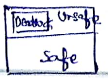
Safe state

- A state is safe, if the system can allocate resources to each process in some order and still avoid a deadlock. It is based on safe sequence → $P_1, P_2, P_3 \cdots P_n$.

BOND WITH THE BEST

Reid & Taylor

Scanned by CamScanner



A safe state is not a deadlock state. A deadlock state is an unsafe state. Not all unsafe states are deadlocks.

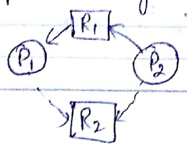
In an unsafe state, the O.S cannot prevent processes from requesting resources such-that a deadlock occurs.

In an unsafe state you can also be in a situation where there might be a deadlock sometime in the future, but it hasn't happened yet becz one or both of the processes haven't actually started waiting.

Process A	Process B	
lock X	lock Y	(State is unsafe)
	unlock Y	
unlock X		(State is safe now, we are lucky)

Resource - Allocation Graph Algorithm :-

In this, we introduce a new type of edge, called a claim edge. A claim edge $P_i \rightarrow R_j$ indicates that process P_i may request resource R_j at some time in future. It is represented by dashed line.



Suppose P_2 Requests R_2. Although R_2 is currently free, we cannot allocate it to P_2, since this action will create a cycle in the graph. A cycle indicates that the system is in an unsafe state. If P_1 requests R_2 and P_1, then a deadlock will occur.

Banker's Algorithm :- RAG is not applicable with a multiple no of instances of resources. Hence Banker's algo is used.

Some data structures must be maintained to implement the banker's algorithm.

Let 'n' be the no. of processes in the system and 'm' be the no of resource types.

- **Available :-** A vector of length m indicates the no. of available resources of each type. If available[j] = K there are 'k' instances of resource type R_j available.
- **Max :-** An n×m defines the max. demand of each process. If Max [i,j] = K, then process P_i may request at most 'K' instances of resource type R_j.
- **Allocation :-** An n×m matrix defines the no. of

BOND WITH THE BEST
Reid & Taylor
Scanned by CamScanner

resources of each type currently allocated to each process. If allocation [i,j] = 'K', then process Pi is currently allocated K instances of resource type Rj.

- <u>Need</u> :- An nxm matrix indicates the remaining resource need of each process. If Need [i,j] = K, the process Pi may need 'K' more instances of resources type Rj to complete its task.

Need [i,j] = Max [i,j] − Allocation [i,j]

<u>Algo</u> :-

1. Let Work and Finish be vectors of length m and n respectively. Initialize Work := Available and Finish [i] = false for i = 1, 2, ... n.

2. Find an i such that both:
   a)- Finish [i] = false
   b)- $Need_i \leq Work$.
   If no such i exists, go to step 4.

3. Work = Work + $Allocation_i$
   Finish [i] = true.
   Go to step 2.

4. If finish [i] = true for all i, then the system is in a safe state.

<u>Ex.</u>

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	3	3	2
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			

We have Need -

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

$< P_1, P_3, P_4, P_2, P_0 >$.

BOND WITH THE BEST

Reid & Taylor

Scanned by CamScanner

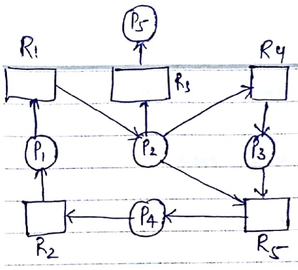
Deadlock Detection :- If deadlock prevention and deadlock avoidence algorithm, doesn't employ then a deadlock situation may occur. Then System must provide:

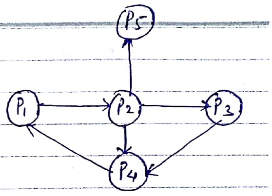
* An algorithm that examines the state of the system to determine whether a deadlock has occured.

* An algorithm to recover from the deadlock.

For Single Instance of Each Resource Type :- For this, we use a wait-for graph which can be obtained by RAG.

An edge from P_i to P_j in a wait-for-graph implies that process P_i is waiting for process P_j to release a resource that P_i needs. An edge $P_i \rightarrow P_j$ exists in a WFG if and only if the corresponding resource-allocation graph contains two edges $P_i \rightarrow R_q$ and $R_q \rightarrow P_j$ for some resource R_q.





Several Instances of a Resource type :- Same as Banker's Algorithm data structures are used-

① Available - A vector of length m indicates the no. of available resources of each type.

② - Allocation - An n×m matrix defines the no. of resources of each type currently allocated to each process.

③ - Request - An n×m matrix indicates the current request of each process. If Request $[i,j] = K$, then process P_i is requesting K more instances of resource type R_j.

Scanned by CamScanner

	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0
P_1	2	0	0	2	0	2			
P_2	3	0	3	0	0	0			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

System is not in deadlocked state due to $\langle P_0, P_2, P_3, P_1, P_4 \rangle$.

If	Request -	P_0	0 0 0	
		P_1	2 0 2	System is deadlocked
		P_2	0 0 1	
		P_3	1 0 0	
		P_4	0 0 2	

Recovery from Deadlock :- Operator deal with Deadlock manually or System recover from the deadlock automatically.
Two options for system recovery-

① - Process Termination —

- Abort all deadlocked processes - This method clearly will break the deadlock cycle, but at a great expense; these processes may have computed for a long-time, and the results of these partial computations must be discarded and probably recomputed later.

- Abort one process at a time until the deadlock cycle is eliminated :-

Resource Preemption :- Three issues need to be addres

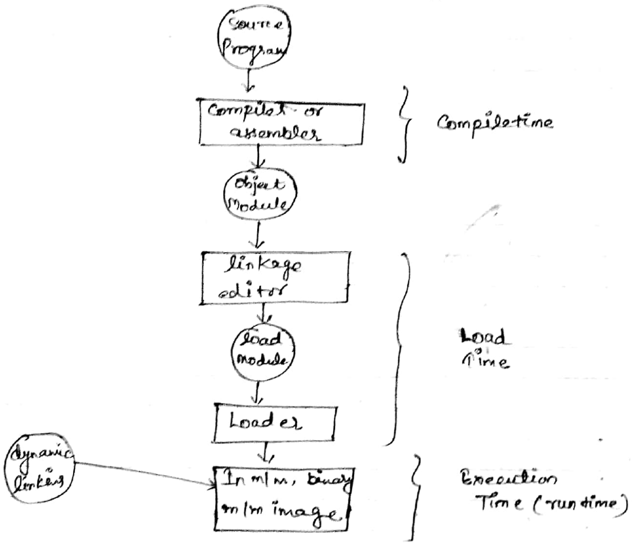
① - **Selecting a victim** - Cost factors includes parameters as the no. of resources a deadlock proces is holding or time.

② - **Rollback** - It is difficult to determine what a saf state is, the simplest solution is a total rollback. Abort the process and then restart it

③ - **Starvation** - A victim cannot be always picked only for a fixed amount of time.

Scanned by CamScanner

Memory Management :- Address Binding -



Compile time - If you know at compile time, where the process will reside in m/m, then absolute code can be generated. like 14000 + 15 ⇒ $\boxed{14015}$

(Binding)
Mapping ⇒ It is a mapping from one address space to another.

Load Time :- If it is not known at compile time where the process will reside in m/m, then the compiler must generate relocatable code. In this case, final binding is delayed until load time. If the starting address changes, we need only to reload the user code to incorporate this changed value.

Execution Time :- If the process can be moved during its execution from one m/m segment to another, then binding must be delayed until run time.

Logical vs. Physical Address Space :-

An address generated by the CPU is commonly referred to as a logical address, whereas an address seen by the m/m unit - that is the one loaded into the m/m - address register of the m/m - is commonly referred to as a physical address.

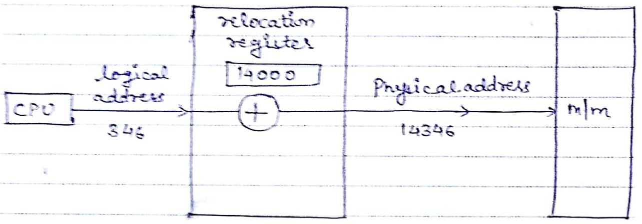
The compile time and load-time address binding methods generate identical logical & physical address. However, the execution - time address binding scheme results in different logical & physical addresses

Scanned by CamScanner

In this case logical address is also called as virtual address.

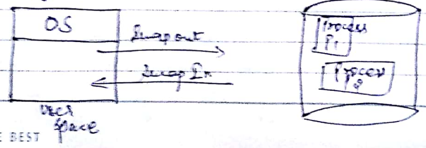
In this set of addresses called logical-address space and physical-address space.

Run-time mapping from V.A to P.A done by (MMU) (m/m mgmt. unit)



Dynamic relocation using a relocation register.

Swapping :- A process needs to be in m/m to be executed. A process, however, can be swapped temporarily out of m/m to a backing store and then brought back into m/m for continued execution



Let us assume that the user process is of size 1 MB and the backing store is a standard hard disk with a transfer rate of 5MB per second.

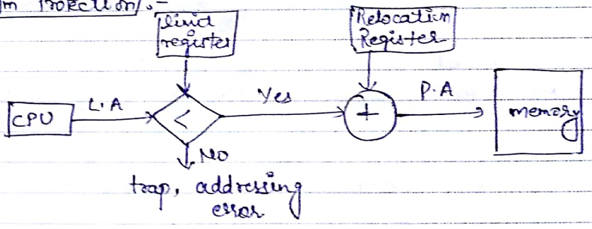
The actual transfer of the 1MB process to or from m/m takes

1000 KB / 5000 KB per sec. = 1/5 sec.

= 200 milliseconds.

Consider avg. latency of 8 milliseconds, so swap time takes 208 ms. So swap in and swap-out total time = 416 ms.

M/m Protection :-



Hardware support for registers

Scanned by CamScanner

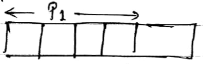
* m/m space should be high
* And access time should be less.

So the conclusion is -

M/m allocation :-

- Contiguous m/m allocation → ⓅⓃ, Ⓟ2, Ⓟ3 — whole process store at main m/m. but problem of external fragmentation.
- Non - Contiguous m/m allocation

We use non-contiguous - pieces of one process store in main m/m whenever the space is free.



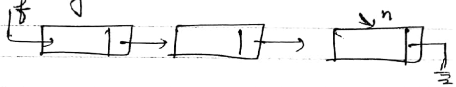
Advantage -
→ access time is very ~~slow~~ fast.

problem -

P_1 = 5KB

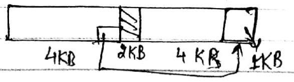
But we cannot allocate the m/m to P_1 becz no contiguous m/m is there. (External fragmentation)

Non - Contiguous Allocation - linked list -



* Access time is very slow. ®

☺

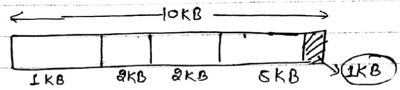


P_1 = 5KB

In Dynamic storage allocation problem - We have 3 schemes for this - (Contiguous m/m allocation)

(fixed size partitioning) -

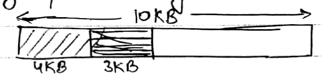
P_1 = 4KB



1KB wastage (Internal fragmentation)

If process comes P_2 = 3KB we can not allocate this to P_2.

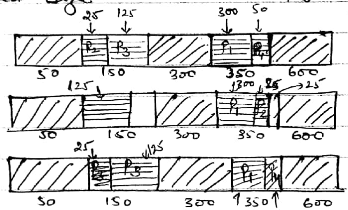
Variable size partitioning -



P_1 = 4KB
P_2 = 3KB

Three schemes for ~~fixed size~~ Space allocation - Not Initial Scenario.

① - First Fit -
② - Best Fit -
③ - Worst Fit — It works good in variable variable size partitioning



Can Not Allocate P_4 (E.F)

P_1 = 300
P_2 = 25
P_3 = 125
P_4 = 50

Reid & Taylor

Scanned by CamScanner

600
491
109

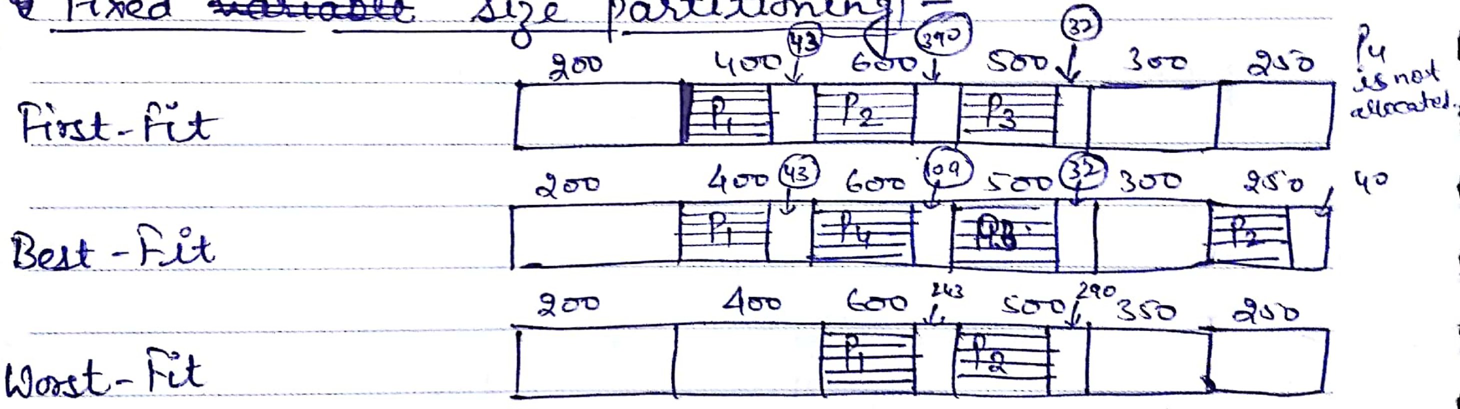
600
210
390

500
468
32

In variable size partitioning, worst fit works best but best fit does not do beez, in best fit we choose best suitable fit block and remaining partition is very slow. (W.F ↑ , B.F ↓)

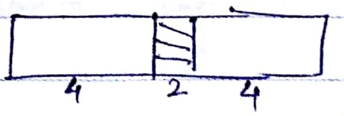
Internal Frag.

Fixed ~~variable~~ Size partitioning ↑ –



$P_1 = 357$
$P_2 = 210$
$P_3 = \boxed{468}$
$P_4 = \boxed{491}$ Not allocated in worst-fit.

E. Frag. depends on problem vector (491).



$P_1 > 9$ KB No ext. f.

If $P_1 > 3$ KB then ①.

$P_1 = 7$ KB then 7KB (e.f.)

Now –