•Methods for Handling Deadlocks

1. Deadlock prevention or avoidance - system is not allowed to get into a deadlocked state.
2. Deadlock detection and recovery - system gets into a deadlocked state and then recovery solutions are implemented.
3. Ignore the problem all together – Deadlocks are handled in a crude fashion such as System Reset! (most real time scenarios this proves to be the best and pragmatic solution! Avoiding the time spent in detecting the point of deadlock or worrying about futuristic combination of requests as with an avoidance setup!
4. This is the approach that both Windows and UNIX take.

•Deadlock Prevention

1. Prevention can be achieved by negating the AND of the necessary conditions discussed earlier.
2. **NOT (A && B && C && D)** implies violation of any one of the four conditions would prevent deadlock from occuring!
3. **NO MUTEX** implies all resources are shared by all processes! In Reality some resources such as printers, tape drives, etc need to be accessed in a NON Shared fashion.
4. **NO HOLD and WAIT**: Start execution only when a process gets all the resources it needs (not efficient!) implies all request calls preceded any release call.
5. Alternatively a new process is allowed to request a new resource only if releases all it has . Both 4 & 5 can result in starvation

•Deadlock Prevention

PREMPTION of resources is allowed!

Wait and Die and Wound – Wait Schemes of Transaction Processing Systems

i) a process that needs additional resources is prempted of other resources it holds (waits and dies..) and later reissued again to gain all resources all over again

ii) a process that needs resources held by other process (say P1); resources are prempted from P1 and assigned to this process. Wounds other processes.

iii) Above solutions are unrealistic with MUTEX and Semaphores (to remember state changes..) possible with CPU registers, memory etc.

•Deadlock Prevention

NO CIRCULAR WAIT

- **One way to realize this is thru a total ordering of resources and requests**
- number all resources, and to require that processes request resources only in strictly increasing (or decreasing) order.
- In other words, in order to request resource Rj, a process must first release all Ri such that i >= j.
- Essence ordering is a one to one function F: R -> N
- F(Tape drive) = 1 ; F(disk drive) = 5 ; F (Printer) = 12
- A Process that holds disk drive cannot request for tape drive or should do so after releasing Tape drive
- Request allowed for Rj only if F(Rj) > F (Ri)
- Release any resource Ri such that F(Ri) >=F(Rj)

- How to prove that above ordering ensures no cyclic wait!
- Proof by contradiction – assume Circular wait holds;
- $P_o, P_1, \ldots P_n$ be the process in the cyclic wait such that P_i waits for R_i held by P_{i+1};
- P_{i+1} is holding R_i which requesting for R_{i+1};
- Implies $F(R_i) < F(R_{i+1})$ – this means
- $F(R_0) < F(R_1) < F(R_2) \ldots < F(R_n) < F(R_0)$
- Which implies $F(R_0) < F(R_0)$ which is not possible. Hence the assumption was wrong.

- Refer to earlier deadlocked code using two mutexes (mutex1 and mutex2)
- Following the ordering among resources principle; lets say $mutex_1 = 5$ and $mutex_2 = 10$;
- Thread or Process 1 lets say gets $mutex_1$ and other thread gets $mutex_2$.
- In this strategy P_1 is allowed to wait for $mutex_2$ but P_2 is not allowed to wait for $mutex_1$;
- hence deadlock is avoided…

Deadlock Avoidance

- Requires more information about each process, leads to low device utilization
- *maximum* number of each resource that a process might potentially use.
- Some scheduler can also take advantage of the *schedule* of exactly what resources may be needed in what order
- If starting a process or granting resource requests may lead to future deadlocks, ; process is just not started
- A resource allocation ***state*** is defined by the number of available and allocated resources, and the maximum requirements of all processes in the system.

Safe State

- system can allocate all resources requested by all processes (up to their stated maximums) without getting into deadlock state
- state is safe if there exists a ***safe sequence*** of processes { P_0, P_1, P_2, ..., P_N } | all of the resource requests for P_i can be granted using the resources currently allocated to P_i and all processes Pj where j < i.
- All processes prior to P_i finish and free up their resources, P_i will be able to finish
- If a safe sequence does not exist, system is in an unsafe state,.
- All safe states are deadlock free, but not all unsafe states lead to deadlocks.

Deadlock Avoidance

Consider a system with 12 tape drives, allocated as follows. Is this a safe state? What is the safe sequence?

	Maximum Needs	Current Needs
P_0	10	5
P_1	4	2
P_2	9	2

- At t0 ; the sequence P_1 P_0 P_2 will be a safe state; available units =12-9=3
- P_1 can ask for 2 more which can be granted and then returned for available units = 5

Now P_0 can be satisfied for its additional 5 units request on completion of which 10 units are available and from there P_2 can be satisfied.

Deadlock Avoidance

- Alternatively assume at some time T_2 P_2 asks for one more unit of the resource and is allocated. Can you find a safe sequence from here?
- Go the initial setup; now the available units = 12-10 = 2. from here only P_1 max request can be entertained and when it finishes available units will be 4.
- From this state neither P_2 nor P_1 can be released the resources they need. This wud be an example for an unsafe state.
- Deadlock results becoz of the additional one unit allocation which had it not been done, then 5 would have been available from which state other processes cud have been completed.

Bankers Algorithm for Deadlock Avoidance

BANKERS algorithms works with the following notations;

Resource vector denoted as R, V denotes available vector; C indicates the Claim matrix while A denotes the current allocation

1. $R_j = V_j + \sum_{i=1}^{n} A_{ij}$, for all j
2. $C_{ij} \leq R_j$, for all i, j
3. $A_{ij} \leq C_{ij}$, for all i, j

New process request is granted only if ;

$$R_j \geq C_{(n+1)j} + \sum_{i=1}^{n} C_{ij} \quad \text{for all } j$$

IIITD&M KANCHEEPURAM

Roll No :

Course No. :

Claim

	R_1	R_2	R_3
P_1	3	2	2
P_2	6	1	3
P_3	3	1	4
P_4	4	2	2

(C)

R_1	R_2	R_3
9	3	6

(R)

Alloc.:-

1	0	0
6	1	2
2	1	1
0	0	2
9	2	5

(A)

$\therefore V = \boxed{0 \mid 1 \mid 1}$ (V)

from this state P_1 can't be run

$\because$ it needs (2 2 2) Not available

2 2 2

P_2 should say:- C

$$\begin{bmatrix} 3 & 2 & 2 \\ 0 & 0 & 0 \\ 3 & 1 & 4 \\ 4 & 2 & 2 \end{bmatrix}$$

A

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix}$$

$V = \boxed{0 \mid 1 \mid 0} +$

6 1 3

$\boxed{6 \mid 2 \mid 3}$

4 0 1

4 0 1

3 2 2

7 2 5

$\therefore \boxed{V = 6,2,3}$

P_1:-

~~0 0 0~~

~~0 0 0~~

3 1 4

4 2 2

~~0 0 0~~

0 0 0

2 1 1

0 0 2

$V = \boxed{7,2,3}$ ✓ 6 2 0

$P_3 \longrightarrow$

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 4 & 2 & 2 \end{matrix} \quad \begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 2 \end{matrix} \qquad V = \begin{matrix} 6 & 2 & 0 \\ 3 & 1 & 4 \\ \hline 9 & 3 & 4 \end{matrix} \; +$$

$\therefore \boxed{V = 9, 3, 4}$

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

P_4 can be run.

$\therefore$ Original state is safe

(9, 3, 6)

(b)

$$\begin{bmatrix} 3 & 2 & 2 \\ 6 & 1 & 3 \\ 3 & 1 & 4 \\ 4 & 2 & 2 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 5 & 1 & 1 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix}$$

C   A 8 2 4

(A)

$\therefore V = \boxed{1 \,|\, 1 \,|\, 2}$

P_2 Needs 1 R1, 1 R3.

res state = Alloc :-

$$\begin{matrix} 1 & 0 & 0 \\ 6 & 1 & 2 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \\ \hline 9 & 2 & 5 \\ \hline 9 & 3 & 6 \end{matrix}$$

can be granted !

$\therefore V = \boxed{0 \,|\, 1 \,|\, 1}$

Stale (A) if P1 [1 R1, 1 R3]

Alloc =

$$\begin{matrix} 2 & 0 & 1 \\ 5 & 1 & 1 \\ 2 & 1 & 1 \\ 6 & 0 & 2 \\ \hline 9 & 2 & 5 \end{matrix}$$

$V = \boxed{0} \quad 1 \quad 1$

potential deadlock ; (P2, P3, P4)

nobody can run until P1 exits / release

$R = R_1 \ldots R_m$ / $V = V_1 \ldots V_m$

$$C = \begin{Bmatrix} C_{11} & \ldots & C_{1m} \\ C_{n1} & \ldots & C_{nm} \end{Bmatrix}$$

$$R = V + \sum_{k=1}^{n} A_{ki}$$

$A_{ki} \le C_{ki}$; $C_{ki} \le R_i$

$P_{n+1} =$ ✓ iff $R_i \geq C_{(n+1)i} +$

$V = \boxed{3\ 2\ 3}$ / 6 2 0