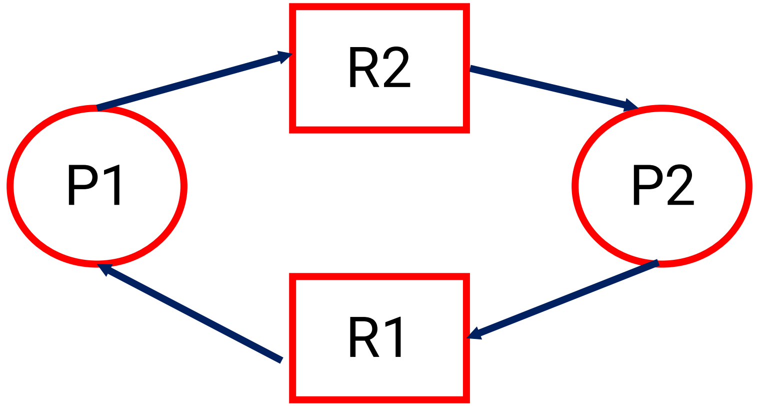
Operating Systems
BCSC 0004

Deadlocks

Deadlock

- In a multiprogramming system, a number of process compete for limited number of resources and if a resource is not available at that instance then process enters into waiting states.
- If a process unable to change its waiting state indefinitely because the resources required by it are held by another waiting process then system is said to be in deadlock.



System Model

- Every process will request for the resources.
- If entertained then, process will use the resources.
- Process must release the resource after use.

Necessary Condition of Deadlock

- **Mutual exclusion:** At least one resource type in the system which can be used in non- shareable mode i.e mutual exclusion (one at a time/ one by one) e. g. CPU time, printer.
- **Hold and wait:** A process is currently holding at least one resource and requesting additional resources which are being held by other processes.
- **No-preemption:** A resource can not be preempted from a process by any other process resource can be released only voluntarily by the process holding it.

Necessary Condition of Deadlock

- **Circular wait:** Each process must be waiting for a resource which is being held by another process, which in turn is waiting for the first process to release the resources

$$\begin{tikzcd} P1 \arrow[r] & P2 \arrow[r] & P3 \arrow[ll, bend left] \end{tikzcd}$$

Deadlock handling Methods

- Prevention: Means design such a system which violate at least one of four necessary conditions of deadlock and ensure independence from deadlock.
- Avoidance: system maintains a set of data using which it takes a decision weather to entertain a new request or not, to be in safe state.
- Detection and recovery: Here we wait until deadlock occurs and once we detect it then recover from it.
- Ignorance: we ignore the problem as if it does not exist.

Deadlock Prevention

❖ This guarantees that there is no deadlock.

❖ Prevention method cannot violate the mutual exclusion condition.

❑ Violate Hold and wait

➢ Conservative approach: Process is allowed to start execution if and only if it has acquired all the resources. (Less efficient, not implementable, easy, deadlock independence)

➢ Do not hold: Process will acquire only desired resources, but before making any fresh request it must release all the resources that it currently hold. (Efficient, implementable)

➢ Wait timeouts: we place a maximum time upto which a process can wait. After which process must release all the holding resources & exit.

Deadlock Prevention

- **Violate No- Preemption**

➢ Forcefully preemption: we allow a process to forcefully preempt the resources holding by other process.

➢ This method may be used by high priority process or system process.

➢ The process which are in waiting state must be selected as a victim instead of process in the running state.

Deadlock Prevention

- **Violate circular wait**

➢ Circular wait can be eliminated by first giving a natural number of every resource

$$F : N \rightarrow R$$

➢ Allow every process to either only in the increasing or decreasing order of the resource number.

➢ If a process require a lesser number (in any of increasing order), then it must first release all the resources larger than required number.

Deadlock Avoidance

- A deadlock avoidance algorithm dynamically examines the resource allocation state.
- The resource allocation state is defined by the number of available and allocated resources and the maximum demands of the processes before allowing that request first.
- We check, if there exist “ some sequence in which we can satisfies demand of every process without going into deadlock, if yes the sequence is called safe sequence” and request can be allowed. Otherwise there is a possibility of going into deadlock.

Banker's Algorithm

- The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety.

- The algorithm for finding out whether or not a system is in a safe state.

1) Let Work and Finish be vectors of length 'm' and 'n' respectively.

Initialize: Work = Available

Finish[i] = false; for i=1, 2, 3, 4....n

2) Find an i such that both

a) Finish[i] = false

b) $Need_i$ <= Work

if no such i exists goto step (4)

3) Work = Work + Allocation[i]

Finish[i] = true

goto step (2)

4) if Finish [i] = true for all i

then the system is in a safe state

Resource-Request Algorithm

1) If $Request_i$ <= $Need_i$

Goto step (2) ; otherwise, raise an error condition, since the process h

as exceeded its maximum claim.

2) If $Request_i$ <= Available

Goto step (3); otherwise, P_i must wait, since the resources are not avai

lable.

3) Have the system pretend to have allocated the requested resources

to process Pi by modifying the state as

follows:

Available = Available – Requesti

$Allocation_i = Allocation_i + Request_i$

$Need_i = Need_i - Request_i$

Considering a system with five processes P_0 through P_4 and three resources of type A, B, C. Resource type A has 10 instances, B has 5 instances and type C has 7 instances. Suppose at time t_0 following snapshot of the system has been taken:

Process	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	

Process	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

Is the system in a safe state? If Yes, then what is the safe sequence?

Applying the Safety algorithm on the given system,

m=3, n=5 Step 1 of Safety Algo

Work = Available

Work =

3	3	2

Finish =

0	1	2	3	4
false	false	false	false	false

For i = 0 Step 2

$Need_0$ = 7, 4, 3

Finish [0] is false and $\overset{7,4,3}{Need_0} > \overset{3,3,2}{Work}$

So P_0 must wait But Need $\leq$ Work

For i = 1 Step 2

$Need_1$ = 1, 2, 2

Finish [1] is false and $\overset{1,2,2}{Need_1} < \overset{3,3,2}{Work}$

So P_1 must be kept in safe sequence

Step 3

Work = $\overset{3,3,2}{Work} + \overset{2,0,0}{Allocation_1}$

Work =

A	B	C
5	3	2

Finish =

0	1	2	3	4
false	true	false	false	false

For i = 2 Step 2

$Need_2$ = 6 , 0, 0

Finish [2] is false and $\overset{6,0,0}{Need_2} > \overset{5,3,2}{Work}$

So P_2 must wait

For i=3 Step 2

$Need_3$ = 0, 1, 1

Finish [3] = false and $\overset{0,1,1}{Need_3} < \overset{5,3,2}{Work}$

So P_3 must be kept in safe sequence

Step 3

Work = $\overset{5,3,2}{Work} + \overset{2,1,1}{Allocation_3}$

Work =

A	B	C
7	4	3

Finish =

0	1	2	3	4
false	true	false	true	false

For i = 4 Step 2

$Need_4$= 4, 3, 1

Finish [4] = false and $\overset{4,3,1}{Need_4} < \overset{7,4,3}{Work}$

So P_4 must be kept in safe sequence

Step 3

Work = $\overset{7,4,3}{Work} + \overset{0,0,2}{Allocation_4}$

Work =

A	B	C
7	4	5

Finish =

0	1	2	3	4
false	true	false	true	true

For i = 0 Step 2

$Need_0$ = 7, 4, 3

Finish [0] is false and $\overset{7,4,3}{Need} < \overset{7,4,5}{Work}$

So P_0 must be kept in safe sequence

Step 3

Work = $\overset{7,4,5}{Work} + \overset{0,1,0}{Allocation_0}$

Work =

A	B	C
7	5	5

Finish =

0	1	2	3	4
true	true	false	true	true

For i = 2 Step 2

$Need_2$ = 6 , 0, 0

Finish [2] is false and $\overset{6,0,0}{Need_2} < \overset{7,5,5}{Work}$

So P_2 must be kept in safe sequence

Step 3

Work = $\overset{7,5,5}{Work} + \overset{3,0,2}{Allocation_2}$

Work =

A	B	C
10	5	7

Finish =

0	1	2	3	4
true	true	true	true	true

Step 4

Finish [i] = true for $0 \leq i \leq n$

Hence the system is in Safe state

The safe sequence is P_1, P_3, P_4, P_0, P_2

What will happen if process P_1 requests one additional instance of resource type A and two instances of resource type C?

$$Request_1 = \begin{array}{c} A \ \ B \ \ C \\ 1, 0, 2 \end{array}$$

To decide whether the request is granted we use Resource Request algorithm

Step 1

$\underset{Request_1}{1, 0, 2} < \underset{Need_1}{1, 2, 2}$

Step 2

$\underset{Request_1}{1, 0, 2} < \underset{Available}{3, 3, 2}$

Step 3

$Available = Available - Request_1$

$Allocation_1 = Allocation_1 + Request_1$

$Need_1 = Need_1 - Request_1$

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

m=3, n=5 — Step 1 of Safety Algo

Work = Available

Work = | 2 | 3 | 0 |

	0	1	2	3	4
Finish =	false	false	false	false	false

For i = 0 — Step 2

$Need_0$ = 7, 4, 3

Finish [0] is false and $Need_0$ > Work (7, 4, 3 > 2, 3, 0)

So P_0 must wait

But Need ≤ Work

For i = 1 — Step 2

$Need_1$ = 0, 2, 0

Finish [1] is false and $Need_1$ < Work (0, 2, 0 < 2, 3, 0)

So P_1 must be kept in safe sequence

Step 3

Work = Work + $Allocation_1$ (2, 3, 0 + 3, 0, 2)

	A	B	C
Work =	5	3	2

	0	1	2	3	4
Finish =	false	true	false	false	false

For i = 2 — Step 2

$Need_2$ = 6 , 0, 0

Finish [2] is false and $Need_2$ > Work (6, 0, 0 > 5,3, 2)

So P_2 must wait

For i=3 — Step 2

$Need_3$ = 0, 1, 1

Finish [3] = false and $Need_3$ < Work (0, 1, 1 < 5, 3, 2)

So P_3 must be kept in safe sequence

Step 3

Work = Work + $Allocation_3$ (5, 3, 2 + 2, 1, 1)

	A	B	C
Work =	7	4	3

	0	1	2	3	4
Finish =	false	true	false	true	false

For i = 4 — Step 2

$Need_4$= 4, 3, 1

Finish [4] = false and $Need_4$ < Work (4, 3, 1 < 7, 4, 3)

So P_4 must be kept in safe sequence

Step 3

Work = Work + $Allocation_4$ (7, 4, 3 + 0, 0, 2)

	A	B	C
Work =	7	4	5

	0	1	2	3	4
Finish =	false	true	false	true	true

For i = 0 — Step 2

$Need_0$ = 7, 4, 3

Finish [0] is false and Need < Work (7, 4, 3 < 7, 4, 5)

So P_0 must be kept in safe sequence

Step 3

Work = Work + $Allocation_0$ (7, 4, 5 + 0, 1 , 0)

	A	B	C
Work =	7	5	5

	0	1	2	3	4
Finish =	true	true	false	true	true

For i = 2 — Step 2

$Need_2$ = 6 , 0, 0

Finish [2] is false and $Need_2$ < Work (6, 0, 0 < 7, 5, 5)

So P_2 must be kept in safe sequence

Step 3

Work = Work + $Allocation_2$ (7, 5, 5 + 3, 0, 2)

	A	B	C
Work =	10	5	7

	0	1	2	3	4
Finish =	true	true	true	true	true

Step 4

Finish [i] = true for $0 \le i \le n$

Hence the system is in Safe state

The safe sequence is P_1, P_3, P_4, P_0, P_2

Hence the new system state is safe, so we can immediately grant the request for process P_1

If a request (3 , 3, 0) by process P4 arrives in the state defined by above can it be granted immediately?

- If a request (0 , 2, 0) by process P0 arrives then check weather it is granted or not? If granted then the new state of the system?

Process	Allocation			Need			Available		
P0	0	3	0	7	2	3	2	1	0
P1	3	0	2	0	2	0			
P2	3	0	2	8	0	0			
P3	2	1	1	0	1	1			
P4	0	0	2	4	3	1			

All five processes are in waiting state as none is able to satisfy the condition

$$Need_i < Available$$