Increase Font Size

(a) *Finish*[*i*] = *false*
(b) *Requesti* £ *Work*
If no such *i* exists, go to step 4.

3. *Work* = *Work* + *Allocationi*
*Finish*[*i*] = *true*
go to step 2.

4. If *Finish*[*i*] == false, for some *i*, 1 £ *i* £ *n*, then the system is in deadlock state.

Moreover, if *Finish*[*i*] == *false*, then *Pi* is deadlocked.

This algorithm requires an order of O(*m* x *n*2) operations to detect whether the system is in deadlocked state. The working of this algorithm can be understood by an example.

**Example of Detection Algorithm**

Consider 5 processes P0 through P4; 3 resource types A, B and C. There are 7 instances of A, 2 instances of B and 6 instances of C.

The snapshot at time T0 is given below:

| | *Allocation* A B C | *Request* A B C | *Available* A B C |
|---|---|---|---|
| P0 | 0 1 0 | 0 0 0 | 0 0 0 |
| P1 | 2 0 0 | 2 0 2 | |
| P2 | 3 0 3 | 0 0 0 | |
| P3 | 2 1 1 | 1 0 0 | |
| P4 | 0 0 2 | 0 0 2 | |

We now simulate the algorithm for the above example.
Initially, Available = (0,0,0); Finish[i] = false for i = 0,1,2,3,4
i = 0
We check if Request0 ≤ Available? Yes
Therefore, Work = Work + Allocation0 =(0,0,0) + (0,1,0) = (0,1,0)
Finish[0] = true , P0 added to safe sequence < P0>

*Allocation* *Request* *Available*

Increase Font Size

| | A B C | A B C | A B C |
|---|---|---|---|
| P0 | 0 1 0 | 0 0 0 | 0 0 0 |
| P1 | 2 0 0 | 2 0 2 | |
| P2 | 3 0 3 | 0 0 0 | |
| P3 | 2 1 1 | 1 0 0 | |
| P4 | 0 0 2 | 0 0 2 | |

Work = (0,1,0);
Is Request1 ≤ Available? No
Since Request1 is not less than Available, check the next process.

*Allocation* *Request* *Available*

| | A B C | A B C | A B C |
|---|---|---|---|
| P0 | 0 1 0 | 0 0 0 | 0 0 0 |
| P1 | 2 0 0 | 2 0 2 | |
| P2 | 3 0 3 | 0 0 0 | |
| P3 | 2 1 1 | 1 0 0 | |
| P4 | 0 0 2 | 0 0 2 | |

Work = (0,1,0);
Is Request2 ≤ Available? Yes
Work = Work + Allocation2 =(0,1,0) + (3,0,3) = (3,1,3) Finish[2] = true , P2 added to safe sequence < P0, P2>

*Allocation* *Request* *Available*

| | A B C | A B C | A B C |
|---|---|---|---|
| P0 | 0 1 0 | 0 0 0 | 0 0 0 |
| P1 | 2 0 0 | 2 0 2 | |
| P2 | 3 0 3 | 0 0 0 | |
| P3 | 2 1 1 | 1 0 0 | |
| P4 | 0 0 2 | 0 0 2 | |

Work = (3,1,3);
Is Request3 ≤ Available? Yes
Work = Work + Allocation3 =(3,1,3) + (2,1,1) = (5,2,4) Finish[3] = true,

P3 added to safe sequence and the safe sequence is now < P0, P2, P3 >

Increase Font Size

*Allocation* *Request* *Available*

| | Allocation | Request | Available |
|---|---|---|---|
| | A B C | A B C | A B C |
| P0 | 0 1 0 | 0 0 0 | 0 0 0 |
| P1 | 2 0 0 | 2 0 2 | |
| P2 | 3 0 3 | 0 0 0 | |
| P3 | 2 1 1 | 1 0 0 | |
| P4 | 0 0 2 | 0 0 2 | |

Work = (5,2,4);
Is Request4 ≤ Available? Yes
Work = Work + Allocation4 =(5,2,4) + (0,0,2) = (5,2,6) Finish[4] = true,
P4 added to safe sequence and the safe sequence is < P0, P2, P3, P4 >

Now, we check again from the beginning all the other processes that were not added to the safe sequence.

*Allocation* *Request* *Available*

| | Allocation | Request | Available |
|---|---|---|---|
| | A B C | A B C | A B C |
| P0 | 0 1 0 | 0 0 0 | 0 0 0 |
| P1 | 2 0 0 | 2 0 2 | |
| P2 | 3 0 3 | 0 0 0 | |
| P3 | 2 1 1 | 1 0 0 | |
| P4 | 0 0 2 | 0 0 2 | |

Work = (5,2,6);
Is Request1 ≤ Available? Yes
Work = Work + Allocation1 =(5,2,6) + (2,0,0) = (7,2,6) Finish[1] = true,
P1 added to safe sequence and the safe sequence now is < P0, P2, P3, P4, P1 >

*Allocation* *Request* *Available*

Increase Font Size

| | | | |
|---|---|---|---|
| | A B C | A B C | A B C |
| P0 | 0 1 0 | 0 0 0 | 0 0 0 |
| P1 | 2 0 0 | 2 0 2 | |
| P2 | 3 0 3 | 0 0 0 | |
| P3 | 2 1 1 | 1 0 0 | |
| P4 | 0 0 2 | 0 0 2 | |

Sequence <P0, P2, P3, P4, P1> now results in *Finish*[*i*] = true for all *i*.

There can be more than one safe sequence, that is there can be correct safe sequences other than <P0, P2, P3, P1, P4>. We have found one safe sequence. Since there is at least one safe sequence, the system is in a safe state. There is no deadlock in the system.

Let process P2 now make an additional request for an instance of resource type C. The Request matrix is changed as shown below, after including the request of an instance of resource type C by process P2.

*Request* A B C

| | |
|---|---|
| P0 | 0 0 0 |
| P1 | 2 0 2 |
| P2 | 0 0 1 |
| P3 | 1 0 0 |
| P4 | 0 0 2 |

Now, let us check if the system will be in a safe state. The deadlock detection algorithm is run again.

*Allocation* *Request* *Available*

| | | | |
|---|---|---|---|
| | A B C | A B C | A B C |
| P0 | 0 1 0 | 0 0 0 | 0 0 0 |
| P1 | 2 0 0 | 2 0 2 | |
| P2 | 3 0 3 | 0 0 1 | |
| P3 | 2 1 1 | 1 0 0 | |
| P4 | 0 0 2 | 0 0 2 | |

Increase Font Size

Initially, Work = Available = (0,0,0); Finish[i] = false for i = 0,1,2,3,4 When i = 0,

Check if Request0 ≤ Available? Yes

Work = Work + Allocation0 =(0,0,0) + (0,1,0) = (0,1,0) Finish[0] = true, P0 is added to safe sequence < P0>

| | *Allocation* | *Request* | *Available* |
|---|---|---|---|
| | A B C | A B C | A B C |
| P0 | 0 1 0 | 0 0 0 | 0 0 0 |
| P1 | 2 0 0 | 2 0 2 | |
| P2 | 3 0 3 | 0 0 1 | |
| P3 | 2 1 1 | 1 0 0 | |
| P4 | 0 0 2 | 0 0 2 | |

Work is now (0,1,0);
Is Request1 ≤ Available? No
Is Request2 ≤ Available? No
Is Request3 ≤ Available? No
Is Request4 ≤ Available? No

Since the request of all the processes cannot be allocated, the system is not in a safe state. Though it possible to reclaim the resources held by process P0, there are insufficient resources to fulfill other processes' requests. Thus, a deadlock exists, consisting of processes P1, P2, P3, and P4.

**Deadlock Detection Algorithm Usage**

When should we invoke the detection algorithm?

This depends on the answers to the following questions.
– How often is a deadlock likely to occur?
– How many processes will be affected by the deadlock when it happens?

If deadlocks occur frequently, then it is necessary to invoke the detection algorithm frequently. If detection is not done, the resources allocated to deadlocked processes will be idle. The number of processes involved in the deadlock cycle may also grow.

Deadlocks occur when a process makes a request that cannot be granted immediately.