**Student Name:** Saurabh Kumar

**Student ID:** 11804508

**Email Address:** [thesaurabh4714@gmail.com](mailto:thesaurabh4714@gmail.com)

**GitHub Link:** https://github.com/thesaurabh4714/CSE316-Operating-System.git

**Description:**

# Deadlock Detection and Recovery

**Deadlock** is a situation where a set of processes are blocked because each processes is holding a resource and waiting for another resource which is already held by some another process.



Three methods to handle deadlock:

1. **Deadlock prevention and avoidance**
2. **Deadlock detection and recovery**
3. **Ignore the problem**

There are certain methods and techniques for deadlock prevention and avoidance. But when the operating system fails to prevent and avoid the deadlock. Then in that case the system will definitely go under deadlock condition. So to get free from deadlock, the operating system periodically checks the system for any deadlock case. If it finds any deadlock then the operating system will recover the system from deadlock using some recovery techniques.

The main task of the OS is detecting the deadlocks. The OS can detect the deadlocks with the help of Resource allocation graph.

In single instanced resource types, if a cycle is being formed within the system then there'll definitely be a deadlock. On the opposite hand, in multiple instanced resource type graph, detecting a cycle isn't barely enough. we've to use the protection algorithm on the system by converting the resource allocation graph into the allocation matrix and request matrix.

In order to recover the system from deadlocks, either OS considers resources or processes.

## For Resource

### Preempt the resource

We can snatch one amongst the resources from the owner of the resource (process) and provides it to the opposite process with the expectation that it'll complete the execution and can release this resource sooner. Well, choosing a resource which is able to be snatched goes to be a small amount difficult.

### Rollback to a safe state

System passes through various states to urge into the deadlock state. The software can rollback the system to the previous safe state. For this purpose, OS has to implement check pointing at every state.

The moment, we get into deadlock, we'll rollback all the allocations to urge into the previous safe state.

## For Process

### Kill a process

Killing a process can solve our problem but the larger concern is to choose which process to kill. Generally, software kills a process which has done thinnest of labor up to now.

### Kill all process

This is not a suggestible approach but will be implemented if the matter becomes very serious. Killing all process will result in inefficiency within the system because all the processes will execute again from starting.

**Constraint Satisfaction:**

Deadlock can arise due to following reasons:

**Mutual Exclusion:** Only one process can use at a time means one or more than one resource are non-sharable.

**Hold and Wait:** A process is holding at least one resource and waiting for resources.

**No Preemption:** A resource cannot be taken form a process unless the process releases the resource.

**Circular Wait:** A set of processes are waiting for each other in circular form.

# Algorithm:

1.  Mark each process that has a row in the Allocation matrix of all zeros.  
2. Initialize a temporary vector W to equal the Available vector.  
3. Find an index *i*such that process *i*is currently unmarked and the *i*th row of Q  
is less than or equal to W. That is, *Q ik*… *Wk*, for 1 … *k*… *m*. If no such row is  
found, terminate the algorithm.  
4. If such a row is found, mark process *i*and add the corresponding row of the  
allocation matrix to W. That is, set *Wk*= *Wk*+ A*ik*, for 1 … *k*… *m*. Return  
to step 3.

**Complexity:**

The complexity of deadlock detection and recovery algorithm is O(n).

**Code:**

# #include<stdio.h> static int mark[20]; int i,j,np,nr; int main() { int alloc[10][10],request[10][10],avail[10],r[10],w[10]; printf("\nEnter the no of process: "); scanf("%d",&np); printf("\nEnter the no of resources: "); scanf("%d",&nr); for(i=0;i<nr;i++) { printf("\nTotal Amount of the Resource R%d: ",i+1); scanf("%d",&r[i]); } printf("\nEnter the request matrix:"); for(i=0;i<np;i++) for(j=0;j<nr;j++) scanf("%d",&request[i][j]); printf("\nEnter the allocation matrix:"); for(i=0;i<np;i++) for(j=0;j<nr;j++) scanf("%d",&alloc[i][j]); /\*Available Resource calculation\*/ for(j=0;j<nr;j++) {

# avail[j]=r[j]; for(i=0;i<np;i++) { avail[j]-=alloc[i][j]; } } //marking processes with zero allocation for(i=0;i<np;i++) { int count=0;  for(j=0;j<nr;j++)    {       if(alloc[i][j]==0)         count++;       else         break;     }  if(count==nr)  mark[i]=1; } // initialize W with avail for(j=0;j<nr;j++)     w[j]=avail[j]; //mark processes with request less than or equal to W for(i=0;i<np;i++) { int canbeprocessed=0;  if(mark[i]!=1) {    for(j=0;j<nr;j++)     {       if(request[i][j]<=w[j])         canbeprocessed=1;       else

# {          canbeprocessed=0;          break;           }      } if(canbeprocessed) { mark[i]=1; for(j=0;j<nr;j++) w[j]+=alloc[i][j]; } } } //checking for unmarked processes int deadlock=0; for(i=0;i<np;i++) if(mark[i]!=1) deadlock=1; if(deadlock) printf("\n Deadlock detected"); else printf("\n No Deadlock possible"); }

**Test Cases:**

**Input**

Enter the no of process: 4  
Enter the no of resources: 5  
  
Total Amount of the Resource R1: 2  
Total Amount of the Resource R2: 1  
Total Amount of the Resource R3: 1  
Total Amount of the Resource R4: 2  
Total Amount of the Resource R5: 1  
  
Enter the request matrix:0 1 0 0 1  
0 0 1 0 1  
0 0 0 0 1  
1 0 1 0 1  
  
Enter the allocation matrix:1 0 1 1 0  
1 1 0 0 0  
0 0 0 1 0  
0 0 0 0 0

**Output**  
 Deadlock detected