**OPERATING SYSTEM DESIGN (5640)**

**PROJECT-2 REPORT**

**Understanding of the given project:**

The purpose of this project is to implement the deadlock detection algorithm in order to check if the system given in a particular state, which is usually represented as General resource allocation graph denoted by G(V,E) which contains a set of processes deadlocked. The system also maintains the track of all resources allocations to the process as well as requests made by the process.

Assumption: The resources used in the system contains only reusable resources.

INPUT given : Adjacency matrix

OUTPUT Expected: List of processes that get deadlocked in the state s.

**Code description:**

In this project, we use the following variables

i,j,k for matrices indices

p-number of processes

r- number of resources

temp- for temporary index

adjmat- adjacency matrix

reqmatrix-request matix

allocmatrix-allocation matrix

t\_allocmatrix- total allocation matrix

t\_res- total resources

alloc\_res- allocated resources

a\_res- available resources

**Detailed Explaination:**

The number of processes, number of resources are given as user inputs.

We used a .txt format file for input and used the file to access the indices in the matrices.

Read the Total resources from the file, indices in adjacency matrices.

Later, we extracted the request as well as allocation matrix from the given adjacency matrix, where in we used a temp variable as there will be change in number of columns and rows respectively.

To calculate the number of resources, we found the transpose of the obtained allocated matrix.

Then we calculated the assigned resources.

We found the number of assigned and the available resources in the present graph state.

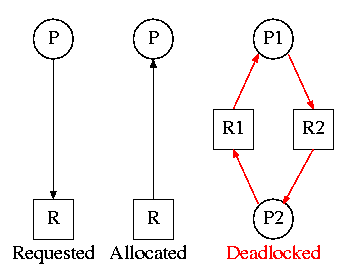
Then we check which process can be completed using the available resources; if the process has the available resources i.e., if the available resources are less than or equal to the needed resources, then the process can be completed and the resources are made free, available for other processes to complete.

If the process is completed, we update the available resources, make sure that it does not check the process completed in the next iteration and the loop continues checking it for all the processes.

If the process cannot be completed as there are insufficient resources; we conclude them to be in deadlock state.

**DEADLOCKS:**

Deadlocks can be defined as the set of blocked processes in which each holds a resource and waits to acquire a resource held by another process.



We consider two different types of resources namely reusable and consumable.

Sharable resources can be used by more than one process at a time.

A consumable resource can only be used by one process, and the resource gets used up.

A serially reusable resource is in between. Only process can use the resource at a time, but once it's done, it can give it back for use by another process.

**Necessary deadlock condition:**

All the below mentioned are the necessary conditions for the deadlock to occur.Deadlocks can be avoided if at least one of the four conditions are not satisfied.

1. **Mutual Exclusion**

Resources shared such as read-only files do not lead to deadlocks but resources, such as printers and tape drives, requires exclusive access by a single process.

1. **Hold and Wait**

In this condition processes must be prevented from holding one or more resources while simultaneously waiting for one or more others.

1. **No Preemption**

Preemption of process resource allocations can avoid the condition of deadlocks, where ever possible.

1. **Circular Wait**

Circular wait can be avoided if we number all resources, and require that processes request resources only in strictly increasing (or decreasing) order.

***Basic Definitions:***

***Sink***: A vertex with an indegree of 0 is called a **source** (since one can *only* leave it) and a vertex with an outdegree of 0

***Isolated node****:* An isolated vertex is a vertex with degree zero; that is, a vertex that is not an endpoint of any edge.

***Path****:* A path is a sequence of edges which connect a sequence of vertices.

***Cycle****:* A sequence of vertices starting and ending at the same vertex, with each two consecutive vertices in the sequence adjacent to each other in the graph

***Reachable set****:* It is the ability to get from one vertex to another within a graph.

***Knot****:* It is a non-empty set of nodes with the property that for all nodes z in K, reachable(z) = K.

***Expediency:*** One resource allocation policy that requires that all satisfiable resource request are granted immediately. In an expedient system, all processes that are requesting resources are blocked.

Limiting the resources to be reusable in our project, we have the following implications:

in a digraph

deadlock -----🡪 cycle

cycle ! --🡪 deadlock

expedient & knot --🡪 deadlock

deadlock !--🡪knot

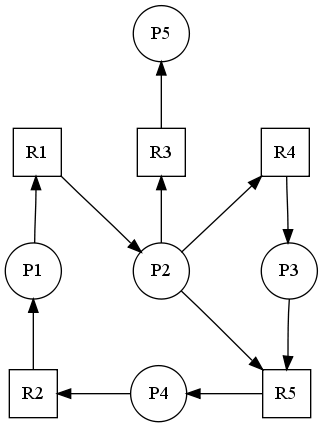
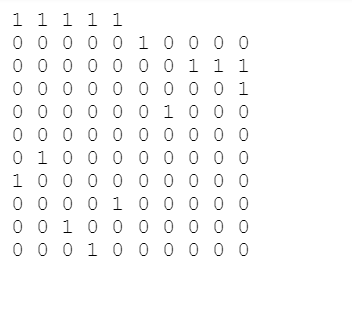
**More efficient deadlock detection:**

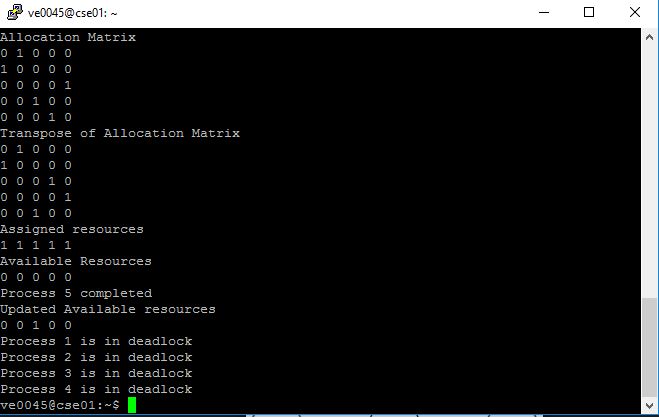
Dreadlocks is an efficient new shared-memory spin lock that actively detects deadlocks. Here, instead of spinning on a Boolean value, each thread spins on the lock owner’s per-thread digest, a compact representation of a portion of the lock’s waits-for graph.

The algorithm uses per-thread transitive closures over portions of the waits-for graph compared to other previous deadlock detection strategies which either maintained an explicit waits-for graph or consisted of elaborate probing mechanisms.

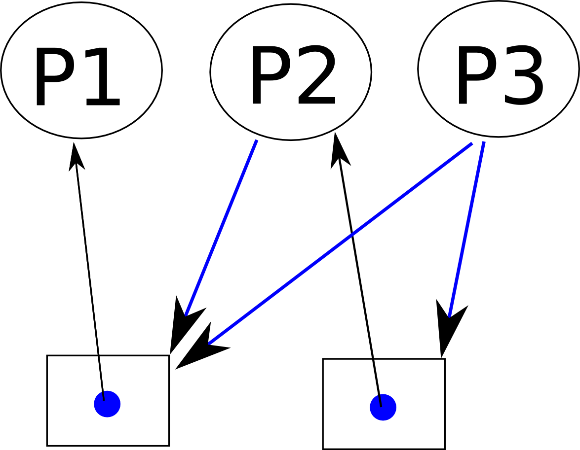
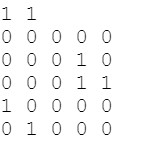
OUTPUTS;

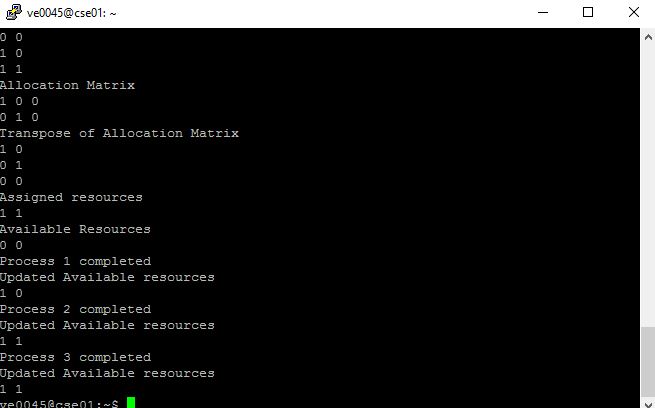
Example 1:

*** ***

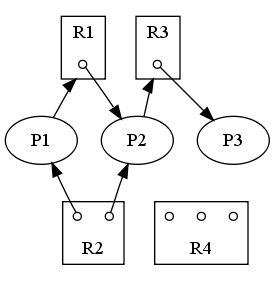
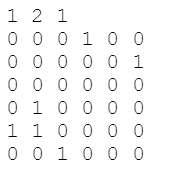
******

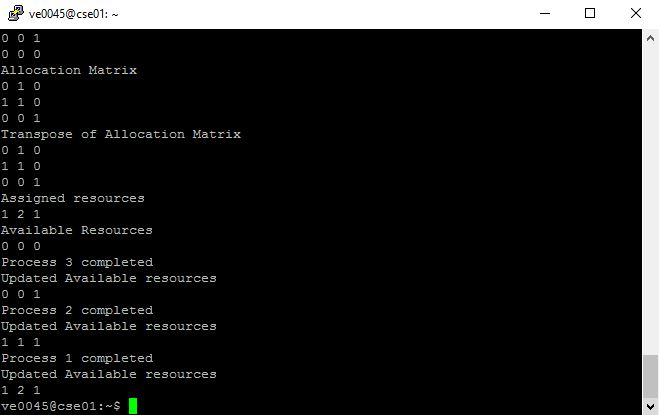
***Example2:***

*** ***

******

***Example 3:***

*** ***

******