CASE STUDY OF DEADLOCK DETECTION IN OPERATING SYSTEM

Submitted MINIPROJECT for OPEARTING SYSTEM LAB (CSL403 )

BY

### Mewada Hitesh Rameshkumar (Group leader) (Roll No. 30)

### Mittal Yash Neeraj (Roll No. 33)

Guide

## Mrs Dipali K Bhole



### Department Of Computer Engineering Shree L. R. Tiwari College Of Engineering

#### Kanakia Park, Mira Road (E), Thane -401 107, Maharashtra.

**Year 2020-21**

iv

# Declaration

We declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.



### HITESH MEWADA (Grp leader)

Roll No.:30



### Yash Mittal

Roll No.:33



Date: 8th May, 2021



CERTIFICATE

This is to certify that the project entitled **“Deadlock detection in operating system ”**is a bonafide work of

**Hitesh Mewada (Grp leader) (Roll no 30)**

**&**

**Yash Mittal (Roll No 33)**

submitted as Mini project of Operating System ,**Computer Engineering**.

#### Signature of Supervisor/Guide Mrs Dipali K Bhole

**\_**



iv

# Table of Contents

Deadlock detection in operating system i

[Declaration ii](#_heading=h.gjdgxs)

[Table of Contents iii](#_heading=h.30j0zll)

[Abstract v](#_heading=h.1fob9te)

1. Introduction 1
   1. Problem statement 1
2. Project Description
3. Diagram 24
4. Features of system/Comparison 30

5 Implementation with code(if any)

6.Conclusion

References 35

iv

# Abstract

In the distributed, system deadlocks is a fundamental problem. A process may request resources in any order, which may not be known in advance and a process can request resource while holding others. Deadlocks can occur if the sequence of the allocations of resources to the processes is not controlled. Fast and efficient deadlock detection is very challenging and difficult task in distributed systems. In this paper distributed deadlock is detected by the distributed control manager. We proposed a distributed deadlock detection algorithm based on the finite automata to detect deadlock in a distributed environment. In this proposed solution we draw the wait for graph for the distributed transaction with the help of finite automata. Our proposed algorithm avoids the transmitting massage to other nodes; it is based on the expansion of an unvisited node in the wait for graph with the help of finite automata. This finite automaton based deadlock detection technique works fast and takes less number of comparisons to detect the deadlock in the wait for graph.

Chapter 1. Introduction

In an operating system, a deadlock occurs when a process or thread enters a waiting state because a requested system resource is held by another waiting process, which in turn is waiting for another resource held by another waiting process.

Chapter 2. Project Description

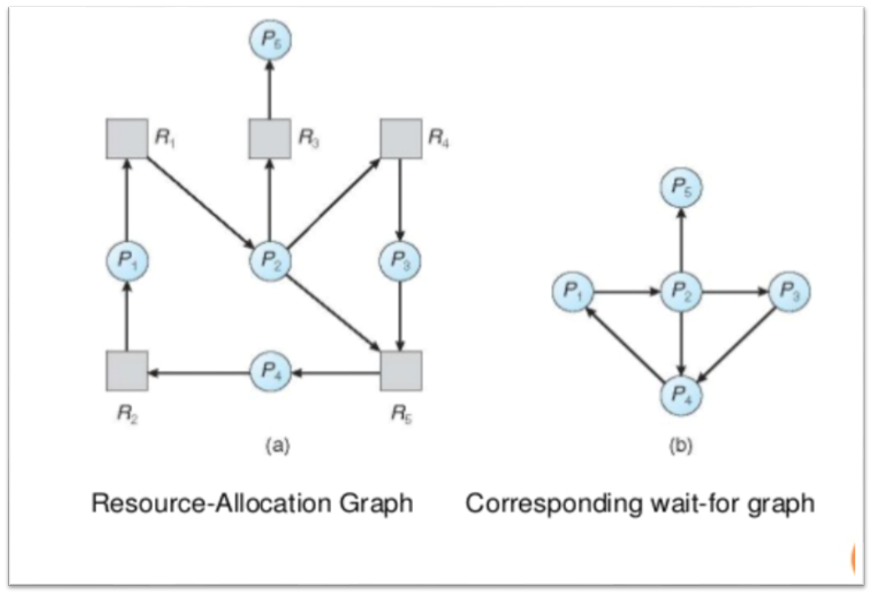
uDeadlock detection is the process of actually determining that a deadlock exists and identifying the processes and resources involved in the deadlock.

u

uOnce a deadlock is detected, there needs to be a way to recover several alternatives exists.

u Temporarily prevent resources from deadlocked processes

Chapter 3 Diagram



Chapter 4 Features of system/Comparison

**Deadlock Recovery**

A traditional operating system such as Windows doesn’t deal with deadlock recovery as it is time and space consuming process. Real-time operating systems use Deadlock recovery.

1. **Recovery methodKilling the process:** killing all the process involved in the deadlock. Killing process one by one. After killing each process check for deadlock again keep repeating the process till system recover from deadlock.
2. **Resource Preemption:** Resources are preempted from the processes involved in the deadlock, preempted resources are allocated to other processes so that there is a possibility of recovering the system from deadlock. In this case, the system goes into starvation.

**Deadlock Avoidance Algorithm/** [**Bankers Algorithm**](https://www.geeksforgeeks.org/operating-system-bankers-algorithm/)**:**

The algorithm employs several times varying data structures:

* **Available –**A vector of length m indicates the number of available resources of each type.
* **Allocation –**An n\*m matrix defines the number of resources of each type currently allocated to a process. Column represents resource and resource represent process.
* **Request –**An n\*m matrix indicates the current request of each process. If request[i][j] equals k then process Pi is requesting k more instances of resource type Rj.

If a system does not employ either a deadlock prevention or [deadlock avoidance algorithm](https://www.geeksforgeeks.org/operating-system-bankers-algorithm-print-safe-state-safe-sequences/) then a deadlock situation may occur. In this case-

* Apply an algorithm to examine state of system to determine whether deadlock has occurred or not.
* Apply an algorithm to recover from the deadlock. For more refer- [Deadlock Recovery](https://www.geeksforgeeks.org/deadlock-detection-recovery/)

Chapter 5. Implementation with code(if any)

#include<stdio.h>

static int mark[20];

int i,j,np,nr;

int main()

{

int alloc[100][100],request[100][100],avail[100],r[100],w[100];

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");

}

output::::::

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

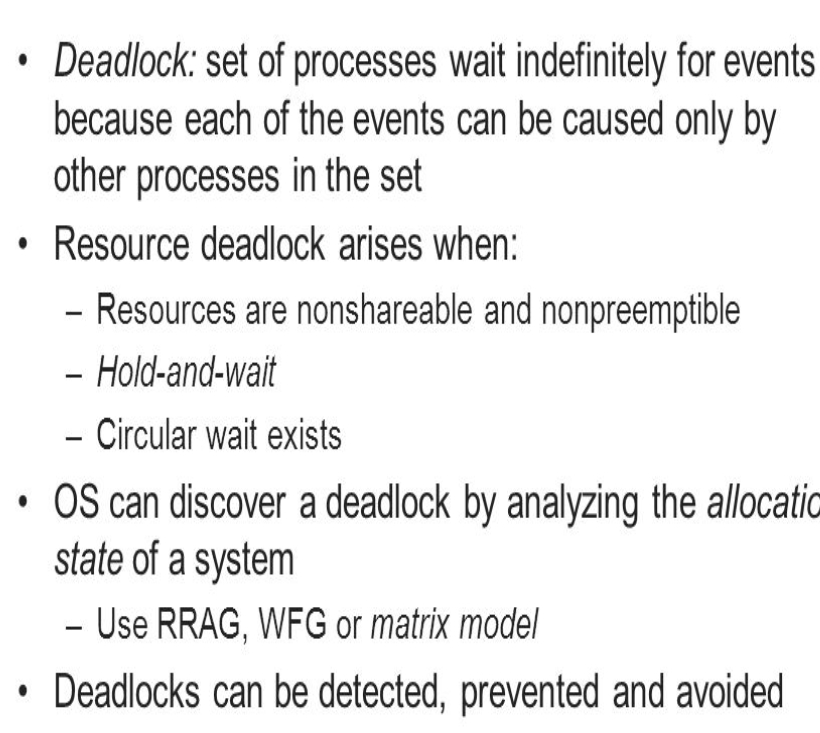
0 0 0 1 0

0 0 0 0 0

Deadlock detected

--------------------------------

Chapter 6. Conclusion



References

[www.operatingsystem.com](http://www.operatingsystem.com)

[www.operatingdeadlocksystem.com](http://www.operatingdeadlocksystem.com)

[www.operatingsystembooks.com](http://www.operatingsystembooks.com)

[www.itsector.com](http://www.itsector.com)

[www.computerstechnology.com](http://www.computerstechnology.com)