[**OS Project Report**](https://www.studocu.com/in/document/peoples-university-india/civil-engineering/scheduling-project-report/45297325?utm_campaign=shared-document&utm_source=studocu-document&utm_medium=social_sharing&utm_content=scheduling-project-report)

**A**

#### MAJOR PROJECT ON

**“DEADLOCKS”**

Submitted in partial fulfillment of the requirement for the degree of

B.TECH

IN

COMPUTER SCIENCE AND ENGINEERING

****

#### ARCHI JAISWAL (RA2211003030369)

#### VIVEK SHARMA (RA2211003030374)

#### UNDER THE SUPERVISION OF

**Mrs. VINAM TOMAR**

**(Assistant Professor)**

**Department of Computer Science & Engineering**

### SRM UNIVERSITY NCR CAMPUS

**OCTOBER - 2023**

# SRM UNIVERSITY NCR CAMPUS MODINAGAR

**Department of Computer Science& Engineering**

# CERTIFICATE

****

This is to certify that Project Report entitled **“DEADLOCK”** submitted by **ARCHI JAISWAL**and **VIVEK SHARMA** has been carried out under my guidance and supervision. The Project is approved for submission towards partial fulfillment for the award of degree of **Bachelor of Technology in Engineering** in **Computer Science & Engineering.**

It is further certified that this work has not been submitted for the award of any other degree or diploma.

**( Mr. VINAM TOMAR)** **(Dr.** )

ASSISTANT PROFESSOR **HOD CSE**

# ACKNOWLEDGEMENT

It gives us immense pleasure to express our deep sense of gratitude to Dr**.** Head of Department of Computer Science & Engineering, whose constant encouragement and valuable guidance have been the source of inspiration in the completion of this work. Without her sustained interest and sound counsel, this work would have not gained the present form.

We are equally thankful to all teaching faculty and ever willing supporting staff members and students of Computer Science & Engineering, School of Research & Technology, Modinagar.

We are also thankful to my family for all their love and affection that helped us to fare well in my task. We thank all those who directly or indirectly helped us during this work.

**Place: SRM UNIVERSITY NCR**

**Date: 30 OCT 2023**

# INDEX

|  |  |  |
| --- | --- | --- |
| SR. NO. | CONTENT | PAGE NO. |
| 1 | PROBLEM STATEMENT | 5 |
| 2 | INTRODUCTION | 7 |
| 3 | DEADLOCK SYSTEM MODEL | 9 |
| 4 | DEADLOCK PREVENTION | 10 |
| 5 | DEADLOCK AVOIDANCE & DETECTION | 11 |
| 6 | BANKER’ ALGORITHM | 12 |

**PROBLEM STATEMENT**

We have a deadlock because neither process can release the resource it is holding until it completes its task, and neither can proceed without the resource the other process is holding. Both processes are effectively “deadlocked,” unable to move forward.

To break the deadlock and free up resources for other processes in this situation, an external intervention, such as the operating system killing one or both processes, would be required. Deadlocks are undesirable in operating systems because they waste resources and have a negative impact on overall system performance and responsiveness. To prevent deadlocks, various resource allocation and [process scheduling algorithms](https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/), such as deadlock detection and avoidance



**ABSTRACT**

The project entitled "DEADLOCK", is basically a program which simulates the following scheduling algorithms such as Banker’s Algorithm

It deals with the following conditions:

1. Mutual Exclusion
2. Hold and Wait
3. Preemption
4. Circular Wait

DEADLOCK is a key concept in computer multitasking, multiprocessing operating system designs. Scheduling refers to the way processes are assigned to run on the available CPUs, since there are typically many more processes running than there are available CPUs. Deadlock deals with the condition where no program wants to attain that condition because here resources are locked by the first process which is entered in the block.

A deadlock occurs when a set of processes is stalled because each process is holding a resource and waiting for another process to acquire another resource. In the diagram below, for example, Process 1 is holding Resource 1 while Process 2 acquires Resource 2, and Process 2 is waiting for Resource 1.

In deadlock avoidance, the request for any resource will be granted if the resulting state of the system doesn't cause deadlock in the system. The state of the system will continuously be checked for safe and unsafe states. In order to avoid deadlocks, the process must tell OS, the maximum number of resources a process can request to complete its execution. The simplest and most useful approach states that the process should declare the maximum number of resources of each type it may ever need. The Deadlock avoidance

algorithm examines the resource allocations so that there can never be a circular wait condition.

Safe and Unsafe States. The resource allocation state of a system can be defined by the instances of available and allocated resources, and the maximum instance of the resources demanded by the

processes.

# INTRODUCTION

**System Model :**

• For the purposes of deadlock discussion, a system can be modeled as a collection of limited resources that can be divided into different categories and allocated to a variety of processes, each with different requirements.

• Memory, printers, CPUs, open files, tape drives, CD-ROMs, and other resources are examples of resource categories.

• By definition, all resources within a category are equivalent, and any of the resources within that category can equally satisfy a request from that category. If this is not the case (i.e. if there is some difference between the resources within a category), then that category must be subdivided further. For example, the term “printers” may need to be subdivided into “laser printers” and “color inkjet printers.”

• Some categories may only have one resource.

• The kernel keeps track of which resources are free and which are allocated, to which process they are allocated, and a queue of processes waiting for this resource to become available for all kernel-managed resources. Mutexes or wait() and signal() calls can be used to control application-managed resources (i.e. binary or counting semaphores. )

• When every process in a set is waiting for a resource that is currently assigned to another process in the set, the set is said to be deadlocked.

**Operations :**

In normal operation, a process must request a resource before using it and release it when finished, as shown below.

1. Request –  If the request cannot be granted immediately, the process must wait until the resource(s) required to become available. The system, for example, uses the functions open(), malloc(), new(), and request ().

2. Use –  The process makes use of the resource, such as printing to a printer or reading from a file.

3. Release – The process relinquishes the resource, allowing it to be used by other processes.

## **MOTIVATION**

In general, there are three approaches to dealing with deadlocks as follows.

1. Preventing or avoiding deadlock by Avoid allowing the system to become stuck in a loop.

2. Detection and recovery of deadlocks, When deadlocks are detected, abort the process or preempt some resources.

3. Ignore the problem entirely.

4. To avoid deadlocks, the system requires more information about all processes. The system, in particular, must understand what resources a process will or may request in the future. ( Depending on the algorithm, this can range from a simple worst-case maximum to a complete resource request and release plan for each process. )

5. Deadlock detection is relatively simple, but deadlock recovery necessitates either aborting processes or preempting resources, neither of which is an appealing option.

6. If deadlocks are not avoided or detected, the system will gradually slow down as more processes become stuck waiting for resources that the deadlock has blocked and other waiting processes. Unfortunately, when the computing requirements of a real-time process are high, this slowdown can be confused with a general system slowdown

## **SCOPE**

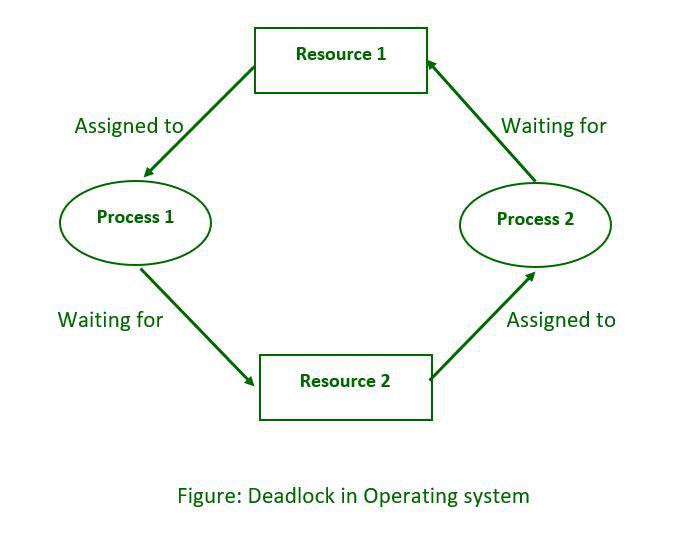
In near future deadlock are moving towards more advancement. It is highly usable and has a bright future in it . Some if the future scope of work are :-

Preemptive deadlock condition like Bankers Algorithms can be explored with similar problem statement and a performance comparison can be me Also, complex conditiom algorithms like deadlock i.e unwanted queue can be explored, which maintains separate class of queues, where each class has its own deadlock need.

## **OBJECTIVE**

**Deadlocks: System Model**

A deadlock occurs when a set of processes is stalled because each process is holding a resource and waiting for another process to acquire another resource. In the diagram below, for example, Process 1 is holding Resource 1 while Process 2 acquires Resource 2, and Process 2 is waiting for Resource 1.



**Deadlock Prevention**:

Deadlocks can be avoided by avoiding at least one of the four necessary conditions: as follows.

Condition-1 : Mutual Exclusion :

• Read-only files, for example, do not cause deadlocks.

• Unfortunately, some resources, such as printers and tape drives, require a single process to have exclusive access to them.

Condition-2 : Hold and Wait :

To avoid this condition, processes must be prevented from holding one or more resources while also waiting for one or more others. There are a few possibilities here:

• Make it a requirement that all processes request all resources at the same time. This can be a waste of system resources if a process requires one resource early in its execution but does not require another until much later.

• Processes that hold resources must release them prior to requesting new ones, and then re-acquire the released resources alongside the new ones in a single new request. This can be a problem if a process uses a resource to partially complete an operation and then fails to re-allocate it after it is released.

• If a process necessitates the use of one or more popular resources, either of the methods described above can result in starvation.

• If a process necessitates the use of one or more popular resources, either of the methods described above can result in starvation.

Condition-3 : No Preemption :

When possible, preemption of process resource allocations can help to avoid deadlocks.

• One approach is that if a process is forced to wait when requesting a new resource, all other resources previously held by this process are implicitly released (preempted), forcing this process to re-acquire the old resources alongside the new resources in a single request, as discussed previously.

• Another approach is that when a resource is requested, and it is not available, the system looks to see what other processes are currently using those resources and are themselves blocked while waiting for another resource. If such a process is discovered, some of their resources may be preempted and added to the list of resources that the process is looking for.

• Either of these approaches may be appropriate for resources whose states can be easily saved and restored, such as registers and memory, but they are generally inapplicable to other devices, such as printers and tape drives.

Condition-4 : Circular Wait :

• To avoid circular waits, number all resources and insist that processes request resources is strictly increasing ( or decreasing) order.

• To put it another way, before requesting resource Rj, a process must first release all Ri such that I >= j.

• The relative ordering of the various resources is a significant challenge in this scheme.

**Deadlock Avoidance**:

The general idea behind deadlock avoidance is to avoid deadlocks by avoiding at least one of the aforementioned conditions.

• This necessitates more information about each process AND results in low device utilization. (This is a conservative approach.)

• The scheduler only needs to know the maximum number of each resource that a process could potentially use in some algorithms. In more complex algorithms, the scheduler can also use the schedule to determine which resources are required and in what order.

• When a scheduler determines that starting a process or granting resource requests will result in future deadlocks, the process is simply not started or the request is denied.

. • The number of available and allocated resources, as well as the maximum requirements of all processes in the system, define a resource allocation state.

**Detection :**

• If deadlocks cannot be avoided, another approach is to detect them and recover in some way.

• Aside from the performance hit of constantly checking for deadlocks, a policy/algorithm for recovering from deadlocks must be in place, and when processes must be aborted or have their resources preempted, there is the possibility of lost work.

**Recovery From Deadlock** :

There are three basic approaches to getting out of a bind:

1. Inform the system operator and give him/her permission to intervene manually.

2. Stop one or more of the processes involved in the deadlock.

3. Prevent the use of resources.

**BANKER’S ALGORITHM**

The banker’s algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for the predetermined maximum possible amounts of all resources, then makes an “s-state” check to test for possible activities, before deciding whether allocation should be allowed to continue.

Why Banker’s Algorithm is Named So?

The banker’s algorithm is named so because it is used in the banking system to check whether a loan can be sanctioned to a person or not. Suppose there are n number of account holders in a bank and the total sum of their money is S. If a person applies for a loan then the bank first subtracts the loan amount from the total money that the bank has and if the remaining amount is greater than S then only the loan is sanctioned. It is done because if all the account holders come to withdraw their money then the bank can easily do it.

It also helps the OS to successfully share the resources between all the processes. It is called the banker’s algorithm because bankers need a similar algorithm- they admit loans that collectively exceed the bank’s funds and then release each borrower’s loan in installments. The banker’s algorithm uses the notation of a safe allocation state to ensure that granting a resource request cannot lead to a deadlock either immediately or in the future.

In other words, the bank would never allocate its money in such a way that it can no longer satisfy the needs of all its customers. The bank would try to be in a safe state always.

Suppose the number of account holders in a particular bank is 'n', and the total money in a bank is 'T'. If an account holder applies for a loan; first, the bank subtracts the loan amount from full cash and then estimates the cash difference is greater than T to approve the loan amount. These steps are taken because if another person applies for a loan or withdraws some amount from the bank, it helps the bank manage and operate all things without any restriction in the functionality of the banking system.

BANKER’S ALGORITHM CODE

#include <stdio.h>

#include <conio.h>

void main() {

int k=0,a=0,b=0,instance[5],availability[5],allocated[10][5],need[10][5],MAX[10][5],process,P[10],no\_of\_resources, cnt=0,i, j;

printf("\n Enter the number of resources : ");

scanf("%d", &no\_of\_resources);

printf("\n enter the max instances of each resources\n");

for (i=0;i<no\_of\_resources;i++) {

availability[i]=0;

printf("%c= ",(i+97));

scanf("%d",&instance[i]);

}

printf("\n Enter the number of processes : ");

scanf("%d", &process);

printf("\n Enter the allocation matrix \n ");

for (i=0;i<no\_of\_resources;i++)

printf(" %c",(i+97));

printf("\n");

for (i=0;i <process;i++) {

P[i]=i;

printf("P[%d] ",P[i]);

for (j=0;j<no\_of\_resources;j++) {

scanf("%d",&allocated[i][j]);

availability[j]+=allocated[i][j];

}

}

printf("\nEnter the MAX matrix \n ");

for (i=0;i<no\_of\_resources;i++) {

printf(" %c",(i+97));

availability[i]=instance[i]-availability[i];

}

printf("\n");

for (i=0;i <process;i++) {

printf("P[%d] ",i);

for (j=0;j<no\_of\_resources;j++)

scanf("%d", &MAX[i][j]);

}

printf("\n");

A: a=-1;

for (i=0;i <process;i++) {

cnt=0;

b=P[i];

for (j=0;j<no\_of\_resources;j++) {

need[b][j] = MAX[b][j]-allocated[b][j];

if(need[b][j]<=availability[j])

cnt++;}

if(cnt==no\_of\_resources) {

op[k++]=P[i];

for (j=0;j<no\_of\_resources;j++)

availability[j]+=allocated[b][j];

} else

P[++a]=P[i];}

if(a!=-1) {

process=a+1;

goto A;}

printf("\t <");

for (i=0;i<k;i++)

printf(" P[%d] ",op[i]);

printf(">");

getch();

}