PROJECT REPORT

CSE- 2005- OPERATING SYSTEM
2018

VIT®
UNIVERSITY
(Estd. u/s 3 of UGC Act 1956)
VELLORE ■ CHENNAI
www.vit.ac.in

DEADLOCK PREVENTION TECHNIQUES

SUBMITTED TO:

SANTHI H.

SUBMITTED BY:

ALOK SINHA	17BCE2380
MANISH PAIKARA	17BCB0141
SUMIT KUMAR SHAH	17BCE2391
BIBEK SINGH	17BCE2393
RAHIL RAIKWAR	17BCE2188

CONTENTS:

MOTIVATION

The resources are not utilized in priority order. In some cases, the processes in a deadlock may wait for infinite time for the resources they need and never end their executions and the resources which are held by them are not accessible to any other process which are depending that resource.

ABSTRACT:

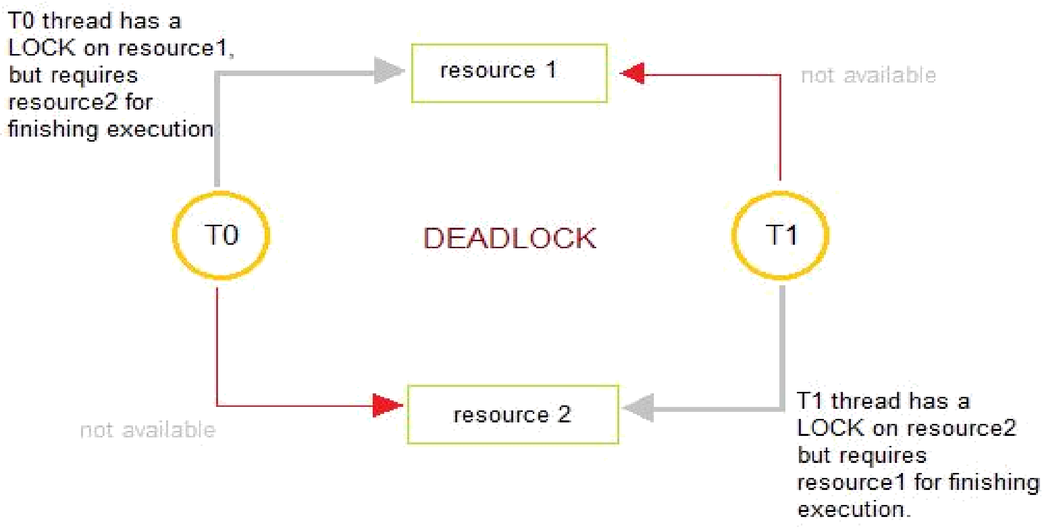
When there is a group of processes waiting for resource which is held by other processes in the same set a deadlock take place. Basically Deadlocks are the set of obstructed processes every one holding a resource and waiting to obtain a resource which is held by some another process. The processes in a deadlock may wait for infinite time for the resources they need and never end their executions and the resources which are held by them are not accessible to any other process which are depending that resource. The existence of deadlocks should be organised efficiently by their recognition and resolution, but may occasionally lead the system to a serious failure. After suggesting the detection algorithm the deadlock is prevented by a deadlock prevention algorithm whose basic step is to allocate the resource only if system will not go in deadlock state. This step prevent deadlock easily. In our project we are going to analyse some deadlock prevention algorithms and will implement them to analyse which one is more suitable for which type of situation.

KEYWORDS:

Deadlock, Conditions for deadlock, Deadlock prevention, Deadlock avoidance, Deadlock detection, Deadlock handling.

INTRODUCTION

A **deadlock** is a situation in which two computer programs sharing the same resource are effectively preventing each other from accessing the resource, resulting in both programs ceasing to function. The earliest computer operating systems ran only one program at a time.[7]


T0 thread has a LOCK on resource1, but requires resource2 for finishing execution
resource 1
not available
T0
DEADLOCK
T1
not available
resource 2
T1 thread has a LOCK on resource2 but requires resource1 for finishing execution.

CONDITIONS FOR DEADLOCK [4]

1. MUTUAL EXCLUSION CONDITION
The resources involved are non-shareable.
Explanation: At least one resource (thread) must be held in a non-shareable mode, that is, only one process at a time claims exclusive control of the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.

2. HOLD AND WAIT CONDITION
Requesting process hold already, resources while waiting for requested resources.
Explanation: There must exist a process that is holding a resource already allocated to it while waiting for additional resource that are currently being held by other processes.

3. NO-PREEMPTIVE CONDITION
Resources already allocated to a process cannot be preempted. **Explanation:** Resources cannot be removed from the processes are used to completion or released voluntarily by the process holding it.

4. CIRCULAR WAIT CONDITION
The processes in the system form a circular list or chain where each process in the list is waiting for a resource held by the next process in the list.

We know that a deadlock is a situation in which two computer programs sharing the same resource are effectively preventing each other from accessing the resource, resulting in both programs ceasing to function.

So in order to prevent Deadlock situation following points are to be implemented:[3]

1-ELIMINATION OF "MUTUAL EXCLUSION" CONDITION
The mutual exclusion condition must hold for non-sharable resources. That is, several processes cannot simultaneously share a single resource. This condition is difficult to eliminate because some resources, such as the tap drive and printer, are inherently non-shareable. Note that shareable resources like read-only-file do not require mutually exclusive access and thus cannot be involved in deadlock.

2-ELIMINATION OF "HOLD AND WAIT" CONDITION
There are two possibilities for elimination of the second condition. The first alternative is that a process request be granted all of the resources it needs at once, prior to execution. The second alternative is to disallow a process from requesting resources whenever it has previously allocated resources. This strategy requires that all of the resources a process will need must be requested at once. The system must grant resources on "all or none" basis. If the complete set of resources needed by a process is not currently available, then the process must wait until the complete set is available. While the process waits, however, it may not hold any resources. Thus the "wait for" condition is denied and deadlocks simply cannot occur. This strategy can

lead to serious waste of resources. For example, a program requiring ten tap drives must request and receive all ten derives before it begins executing. If the program needs only one tap drive to begin execution and then does not need the remaining tap drives for several hours. Then substantial computer resources (9 tape drives) will sit idle for several hours. This strategy can cause indefinite postponement (starvation). Since not all the required resources may become available at once.

3-ELIMINATION OF “NO-PREEMPTION” CONDITION

The nonpreemption condition can be alleviated by forcing a process waiting for a resource that cannot immediately be allocated to relinquish all of its currently held resources, so that other processes may use them to finish. Suppose a system does allow processes to hold resources while requesting additional resources. Consider what happens when a request cannot be satisfied. A process holds resources a second process may need in order to proceed while second process may hold the resources needed by the first process. This is a deadlock. This strategy require that when a process that is holding some resources is denied a request for additional resources. The process must release its held resources and, if necessary, request them again together with additional resources. Implementation of this strategy denies the “no-preemptive” condition effectively. High Cost When a process release resources the process may lose all its work to that point. One serious consequence of this strategy is the possibility of indefinite postponement (starvation). A process might be held off indefinitely as it repeatedly requests and releases the same resources.

4-ELIMINATION OF “CIRCULAR WAIT” CONDITION

The last condition, the circular wait, can be denied by imposing a total ordering on all of the resource types and than forcing, all processes to request the resources in order. (increasing or decreasing). This strategy impose a total ordering of all resources types, and to require that each process requests resources in a numerical order (increasing or decreasing) of enumeration. With this rule, the resource allocation graph can never have a cycle. For example, provide a global numbering of all the resources, as shown

1 ≡ Card reader
2 ≡ Printer
3 ≡ Plotter
4 ≡ Tape drive
5 ≡ Card punch

Now the rule is this: processes can request resources whenever they want to, but all requests must be made in numerical order. A process may request first printer and then a tape drive (order: 2, 4), but it may not request first a plotter and then a printer (order: 3, 2). The problem with this strategy is that it may be impossible to find an ordering that satisfies everyone.

AIM

Deadlock Prevention: Preventing deadlocks by constraining how requests for resources can be made in the system and how they are handled (system design). The goal is to ensure that at least one of the necessary conditions for deadlock can never hold.

PROJECT SCOPE / APPLICABILITY

This thesis strives to develop algorithms that minimize the messages as well as their length for detecting deadlocks without compromising other performance measures such as deadlock duration and deadlock resolution time.

1.ONE-SHOT ALGORITHM
2.MULTISHOT ALGORITHM
3.HIERARCHICAL ALGORITHM
4.HIERARCHICAL ALGORITHM (WITH QUEUE)

OBJECTIVE

- To develop a description of deadlocks, which prevent sets of concurrent process from completing their tasks.
- To present a number of different methods for preventing or avoiding deadlocks in a computer system.

SOFTWARE SPECIFICATION

C++

LITERATURE SURVEY+ METHODOLGY

S. No	Research Paper	Researcher Name	Algorithm	Existing System	Performance Measures
1.	Deadlock Resolution Techniques: An Overview	Pooja Chahar, Surjeet Dalal	VGS Algorithm for Deadlock Resolution: This algorithm is based on the mutual cooperation of the transactions.	Mendivil"s algorithm a process with the lowest priority is aborted.	Deadlock detection is the process of actually determining that a deadlock exists and identifying the processes and resources involved in the deadlock. For this reason several issues must be carefully considered. 1) Aborts are more expensive than waits. 2) Unnecessary aborts result in wasted system resources. 3) Optimal concurrency requires that the number of aborted transactions be minimized
2.	Efficient Deadlock Avoidance for	Jeremy D. Buhler Kunal Agrawal	Destination-Tagged	we describe some of the background for filtering applications,	We will concentrate on describing the algorithms for the corner sources. We will

	Streaming Computation with Filtering	Peng Li Roger D. Chamberlain	Propagation Algorithm	deadlock avoidance, and SP-DAGs.	describe all the algorithms for some ui, where ui is a corner node on the left path of the ladder. Analogous algorithms can be derived for nodes on the right path.
3.	VGS Algorithm - an Efficient Deadlock Resolution Method	Kunwar Singh Vaisla Menka Goswam	VGS ALGORITHM FOR DEADLOCK RESOLUTION	WGF(weight for graph)	In a distributed system if deadlock is detected at a site, then the site coordinator can apply VGS algorithm to resolve the deadlock. This algorithm is based on the mutual cooperation of the transactions
4.	A Study of Distributed Deadlock Handling Techniques	Garima Rana, Parveen Kumar, Kanchan Choudhary	A DIFFUSION COMPUTATION BASED ALGORITHM	1. Path pushing algorithm 2. Edge chasing algorithm	Initiate a diffusion computation for a blocked process Pi. When a blocked process Pk receives a query (i, j, k), a process Pk receives a reply (i, j, k).
5.	Distributed Deadlock Handling	Dilshana Khurshid, Kanchan Choudhary	A GLOBAL STATE DETECTION BASED ALGORITHM	1. Diffusion computation algorithm 2. Global state detection algorithm	the algorithm regulates the granting of requests to check for various deadlocks. The first phase terminates after the second phase has been terminated because the first phase is nested within the second phase. Now the, the algorithm by Wang et al. also includes two phases. In the first phase a place description of distributed WFG is recorded whereas in the second phase, the static WFG recorded in the initial phase is reduced to detect any deadlocks. This algorithm has one major feature that is, both the phases occur simultaneously
6.	Deadlock Prevention Algorithm in Grid Environment	Deepti Malhotra	DEADLOCK PREVENTION ALGORITHM WITH NO PREEMPTION.	Replica Based local Deadlock Prevention	The Algorithm that we are proposing and trying to implement is based on the idea of No-preemption for deadlock prevention, existence of deadlocks in a smaller network could be overcome easily, but if we are dealing with a Grid

					environment, it could be technically and economically infeasible to deal with such a situation.
7.	Deadlock Prevention in Process Control Computer System	Manisha Mohanty, Prerna Kumara	PETRI NETS ALGORITHM FOR THE PREVENTION OF DEADLOCK.	DIRECTED GRAPH FOR WAIT RELATION AND RESTRICTION MATRIX	BASICS OF PETRI NETS Petri nets are a graphical and mathematical modelling tool applicable to many systems. It is a promising tool for describing and studying information processing systems that are characterized as being concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic.
8.	A SURVEY OF DISTRIBUTE D DEADLOCK DETECTION ALGORITH MS	Ahmed K. Elmagarmid	Menasce's Scheme [MENASCE79]	Algorithms that belong to the first category pass information about transaction requests to maintain global wait-for-graph.	This algorithm was the first to use a condensed transaction-wait-for graph (TWF) in which the vertices represent transactions and edges indicate dependencies between transactions. In this graph, an edge (Ti,T 3) exists is T 1 is blocked on T3 and must wait for T3 to release the resource.
9.	Robust Deadlock Avoidance for Sequential Resource Allocation Systems with Resource Outages	Spyros Reveliotis and Zhennan Fei	Based on the Sequential Logic Requirement Vectors	Process Based on the Sequential Logic	Single-Unit: Each stage defined by a linear sequence requires a single unit of stages from a single resource Single-Type: Each stage requires an arbitrary number of units, but all from a single resource
10.	Deadlock Avoidance Based on Banker's Algorithm for FMS	Xu Gang, Wu Zhiming	Improvement of Banker algorithm	Banker's Algorithm	The Dijkstra Banker's algorithm operates commonly in the Deadlock-Avoidance of the Operating System's process Safe State, at this state there is at least one resource allocation sequence that can make all

					processes finish safely, and no deadlock produce.
11	Deadlock-Free Typestate-Oriented Programming	Luca Padovani	Refined TypeState-Oriented Programming	TypeState-Oriented Programming	Refinement of TSOP for concurrent objects guarantees programs which follow the protocol of the objects they use to be deadlock free. Approach fully compositional and scalable. This provides join patterns and high-level concurrency abstraction with which it is easier to synchronize complex programs in declarative form.
12	Reliable Resource Provisioning Using Bankers' Deadlock Avoidance Algorithm in MEC for Industrial IoT	Emeka E. Ugwuanyi Saptarshi Ghosh Muddesar Iqbal Tasos Dagiuklas	Lamport's algorithm Chandy-Misra-Hass algorithm Parallel deadlock detection algorithm Detection in heterogeneous systems Load balancing Methods Banker's Algorithm	Banker's Algorithm	Resource provisioning algorithm for deadlock avoidance for multi-access edge platform was presented with the aim of maintaining reliable network for IoT devices and performed well on simulations tests.
13	Petri Nets and Deadlock-Free Scheduling of Open Shop Manufacturing Systems	Gonzalo Mejía ; Juan Pablo Caballero-Villalobos ; Carlos Montoya	Petri net class extended to S3R nets to handle open shop systems	Petri net	The algorithm provides necessary condition for deadlock free environment based on properties of particular structural siphon,
14	Predicting the Time to Migrate Into Deadlock Using a Discrete Time Markov Chain	André B. Bondi	Markov Chain analysis in FCFS	FCFS	Uses combination of request and other most likely reason for cause of deadlock and uses Markov chain analysis to predict time for deadlock and prevent them from happening
15	Deadlock control of multithreaded software based on Petri nets: A brief review	M. D. Gan ; Z. J. Ding ; S. G. Wang ; W. H. Wu ; M. C. Zhou	Iterative control of ordinary Gadara nets via Mixed Integer Mapping and petri nets	Gadara nets Petri Nets	iterative control of general Gadara nets, iterative control of ordinary Gadara nets via Mixed Integer Programming (MIP) and iterative control of ordinary Gadara nets via Boolean satisfiability formulation (SAT)
16	Deadlock Property Analysis of	Wei Liu; Lu Wang; Yuyue Du; Maozhen Li	analysis of deadlocks of process nets	Petri net model	Analysis of deadlocks of process nets, deadlocks of concurrent programs are

	Concurrent Programs Based on Petri Net Structure				concluded. For an important subclass, deadlock property of concurrent programs is determined based on the deadlock property and the static structure of process nets
17	Deadlock detection, prevention, and avoidance for automated tool sharing systems	N.Z. Gebraeel ; M.A. Lawley	Polynomial Algorithm	FCFS	Automated tool sharing systems provide a technological response to high cost of flexible manufacturing systems by allowing different machines to use same tools without any problems and without deadlock
18	A siphon-based deadlock prevention policy for flexible manufacturing systems	Yi-Sheng Huang ; Jenn-Huei Lin ; Jyh-Tsong Lin	Siphon based algorithm	Petri nets	Approach by adding two kinds of control places called ordinary control places and weighted control places to the original model to prevent siphons from being unmarked.
19	Approaches for Deadlock Detection and Deadlock Prevention for Distributed systems5	Gupta Dhiraj and Gupta V.K.	Path-pushing algorithm Edge-chasing Algorithm Distributed deadlock prevention		distributed deadlock, deadlock dictation and prevention. In this way, the communication overhead of the deadlock detection procedure is reducing.
20	DEADLOCK PREVENTIO N AND DETECTION IN DISTRIBUTE D SYSTEMS	NAVIN KUMAR	Ho-Ramamoorthy Algorithm; Diffusion Computation Based Algorithm; Menasce-Muntz Algorithm	Banker’s Algorithm	This paper does comparative study and shows methods to decrease time lag caused by deadlock detection in real world systems

<u>ONE-SHOT ALGORITHM [21]</u>

In one shot algorithm one protocol requires each process to request and be allocated all its resources before it begins execution. Given a request from process P for resources R1, R2, ..., Run, the resource manager follows these rules

```
if process P has ever acquired any resources before, then refuse the request
else if any resource R1, ... Rn does not exist then
refuse the request
else
{if any resource R1, ... Rn is not free, then wait until all resources R1, ... Rn are free end if
grant process P exclusive access to resources R1, ... Rn} end
```

EXAMPLE: -

Suppose that a person needs both a knife and fork to eat.
Person P1 needs knife and fork and person P2 needs knife and fork.
Person P1 requests knife and fork, person P2 also requests knife and fork.
Only one of P1 or P2 will be granted the resources.
Suppose it is P1. P2 is forced to wait.
Person P1 uses the knife and fork until finished.
Person P1 releases the knife and fork.

Since the resources that P2 was waiting for are free, P2 is granted both the knife and fork.
Person P2 uses the knife and fork until finished. Person P2 releases the knife and fork.

MULTISHOT ALGORITHM [21]

In multishot or **repeated one shot** algorithm, an alternative protocol allows a process to request resources only when the process has none. A process may request some resources and use them. Before it can request any additional resources, however, it must release all the resources that it is currently allocated.

Given a request from process P for resources R1, R2, ..., Rn, the resource manager follows a similar rule to that for one-shot

```
if process P currently has any resources,
then refuse the request
else if any resource R1, ... Rn, does not
exist, then refuse the request else

 { if any resource R1, ... Rn is not free,
then wait until all resources R1, ... Rn are free end if
grant process P exclusive access to resources R1, ...
Rn }
 end if
```

If a process P wants to request resources while holding resources, they follow these steps:

P frees all resources being held P requests all resources previously held plus the new resources it wants to acquire.

HIERARCHICAL ALGORITHM [21]

Assume the resources have unique priorities (i.e., all priorities are different). Given a request from process P for resource R, the resource manager follows these rules:

```
if process P currently has any resources with equal or higher priority than
resources R, then
 refuse the request
 else if resource R1 does not exist, then
 refuse the request else
{
 if the resource R is not free, then wait until resource R is free
 end if
 grant process P exclusive access to resources R
} end if
```

EXAMPLE:

Suppose that a person needs both a knife and fork to eat. Knife is given priority 2 and fork priority 1, assuming that zero is the highest priority. Person P1 needs knife and fork and person P2 needs knife and fork. Person P1 requests knife first because it is lower priority and then fork. Person P2 also requests knife and then fork. Only one of P1 or P2 will be granted the knife. Suppose it is P1. Person P2 will wait for the knife to be free. Then only P1 will request the fork. The request will be granted. Person P1 will use the knife and fork until finished. Person P1 will release the knife and fork. Since person P2 is waiting for the knife, the request can now be granted. Then Person P2 will immediately request the fork and this request will also be granted. Person P2 will use the knife and fork until finished. Person P2 will release the knife and fork.

HIERARCHICAL ALGORITHM (WITH QUEUE)[21]

In hierarchical algorithm, each process can request resources only in an increasing order of enumeration. If several instances of the same resource type are needed, a single request for all of them must be issued.

An interesting variation on the Hierarchical Algorithm can be created by ensuring that the waiting done by the processes is done on a series of queues, one per resource. Given a request from process P for resource R, the resource manager follows these rules:

```
if process P currently has any resources with equal or higher priority than resources R, then
refuse the request
 else if resource R does not exist, then
 refuse the request
else {
if the resource R is not free, then put process P in a queue waiting for resource R and when process P reaches the front of the queue, grant process P exclusive access to resource R
end if
 }
 end if
```

Example:

Suppose that a person needs both a knife and fork to eat. Knife is given priority 2 and fork priority 1, assuming that zero is the highest priority. Person P1 needs knife and fork and person P2 needs knife and fork. Person P1 requests knife first because it is lower priority and then fork. Person P2 also requests knife and then fork. Only one of P1 or P2 will be granted the knife. Suppose it is P1. Person P2 will be put in the queue for the knife. Then only P1 will request the fork. The request will be granted. Person P1 will use the knife and fork until finished. Person P1 will release the knife and fork. Since person P2 is at the front of the queue waiting for the knife; person P2 will be granted the knife. Then Person P2 will immediately request the fork and this request will be granted. Person P2 will use the knife and fork until finished. Person P2 will release the knife and fork.

WORK IMPLEMENTATION

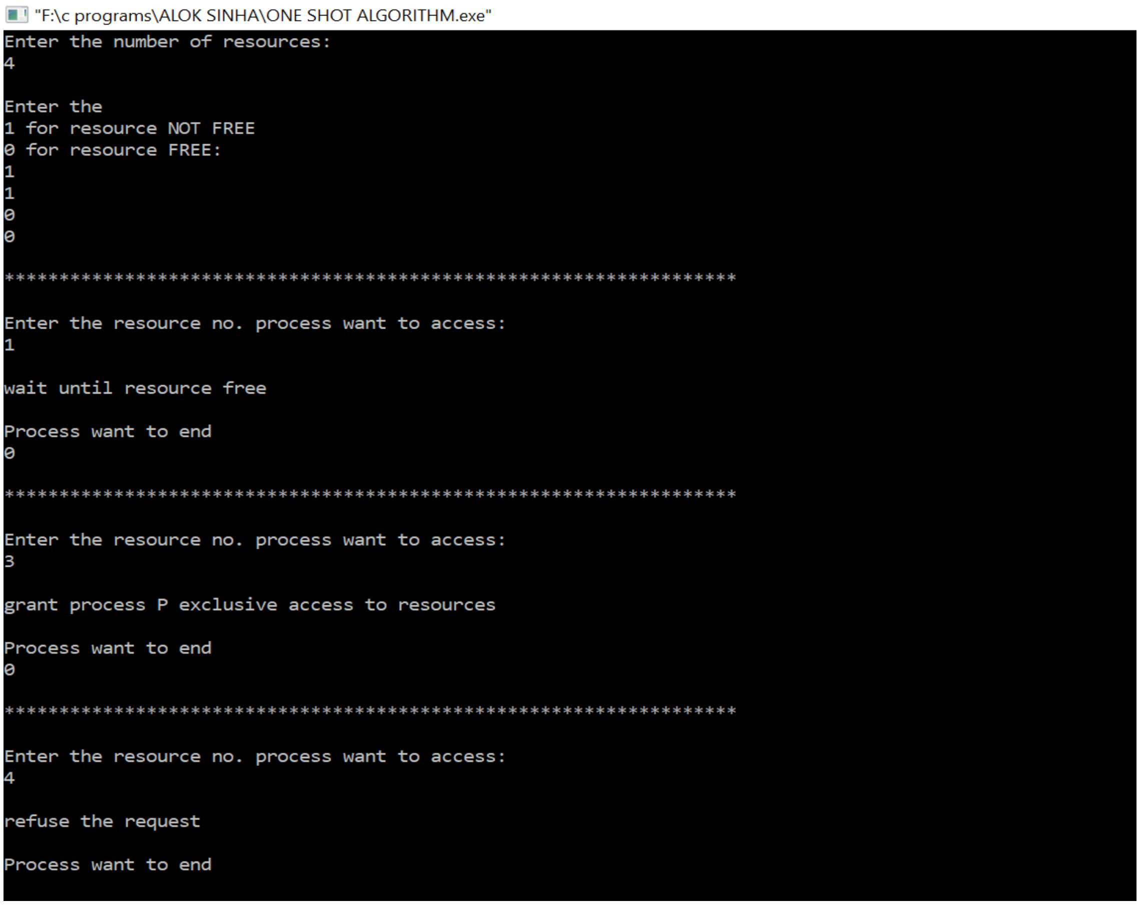
In this project, our aim is to implement the deadlock prevention algorithms and analyse the differences between the algorithms in a suitable way. Deadlock prevention is a very serious work for an operating system. If even a small deadlock occur then it can cause whole system failure.
In our project we will be analysing and implementing the deadlock prevention algorithms

i) We will enter all the resources required by the processes.
ii) The request will automatically be rejected if the demand of resources is greater than the no of resources present.
iii) Then we will find if safe sequence is possible or not.
iv) Then we will implement all the algorithm, and find the arrangement if the resources to processes in such way that the deadlock is prevented.
v) After that resources is allocated to the processes.

EXPERIMENTS/RESULTS

ONE SHOT ALGORITHM

```
#include<iostream>
using namespace std;
int main (){
        int n,m,att=0;
        cout<<"Enter the number of resources:\n";
        cin>>n;
        int res[n];

        cout<<"\nEnter the \n1 for resource NOT FREE \n0 for resource
        FREE:\n"; for(int i=0;i<n;i++){
                cin>>res[i];
        }

        int choice=0;

        while(choice==0){
        cout<<"\n*******************************************************
************\n";
        int count=0,q=0,l=0;
        cout<<"\nEnter the resource no. process want to access:\n";
        cin>>m;
        if(att!=0 ){
        cout<<"\nrefuse the request\n";
        }
        else if(m>n || m<1)
        cout<<"\nResource does not exists\n";
        else{
                if(res[m-1]!=0){
                        cout<<"\nwait until resource free \n";

                                res[m-1]=0;


                }
                else if(res[m-1]==0){
                        cout<<"\ngrant process P exclusive access to resources\n";
                        att=1;
                        res[m-1]=1;

                }
        }

        cout<<"\nProcess want to end \n";
        cin>>choice;
}
return 0;
}
```

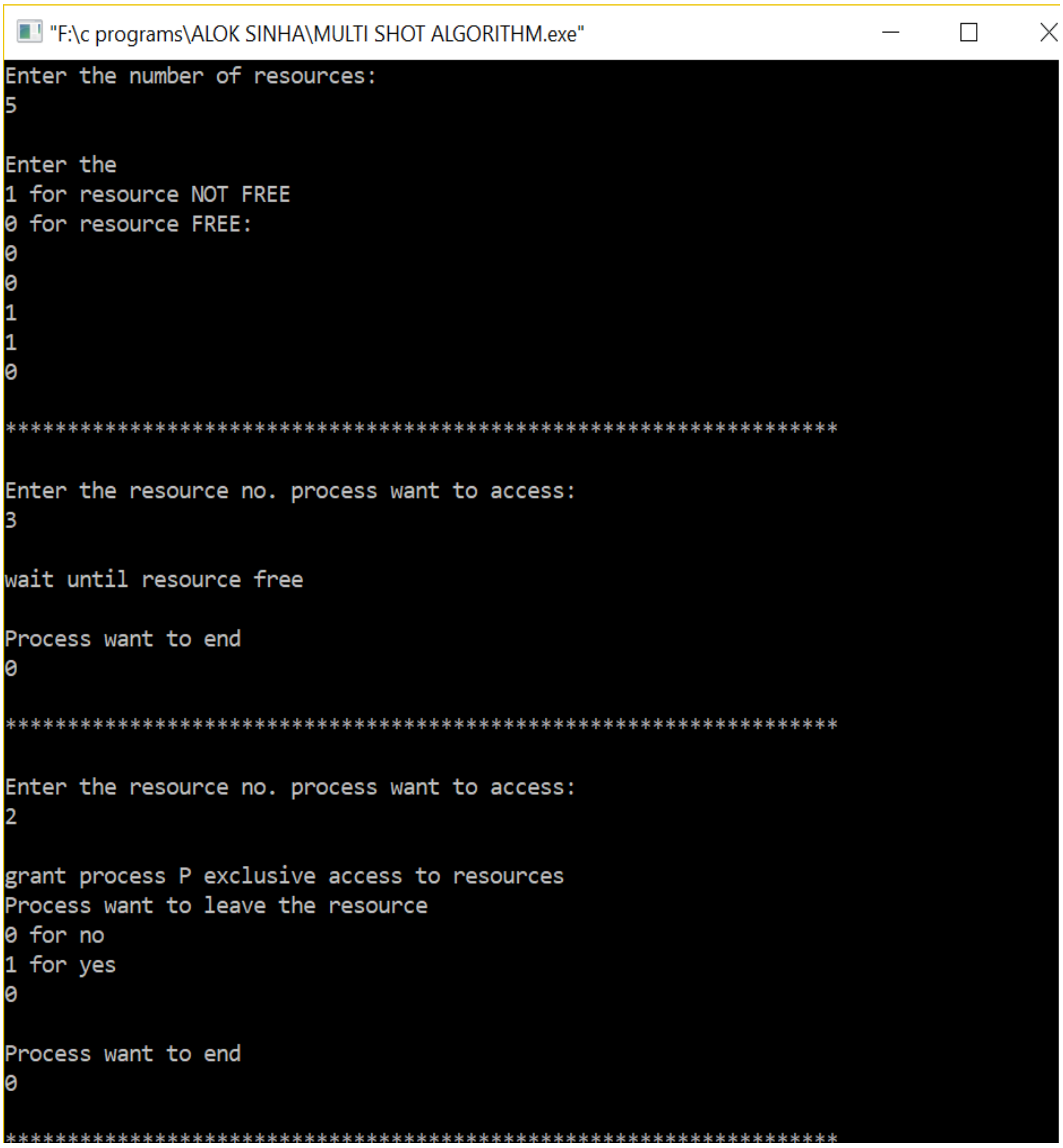


```
"F:\c programs\ALOK SINHA\ONE SHOT ALGORITHM.exe"
Enter the number of resources:
4

Enter the
1 for resource NOT FREE
0 for resource FREE:
1
1
0
0

********************************************************************

Enter the resource no. process want to access:
1

wait until resource free

Process want to end
0

********************************************************************

Enter the resource no. process want to access:
3

grant process P exclusive access to resources

Process want to end
0

********************************************************************

Enter the resource no. process want to access:
4

refuse the request

Process want to end
```

MULTISHOT ALGORITHM

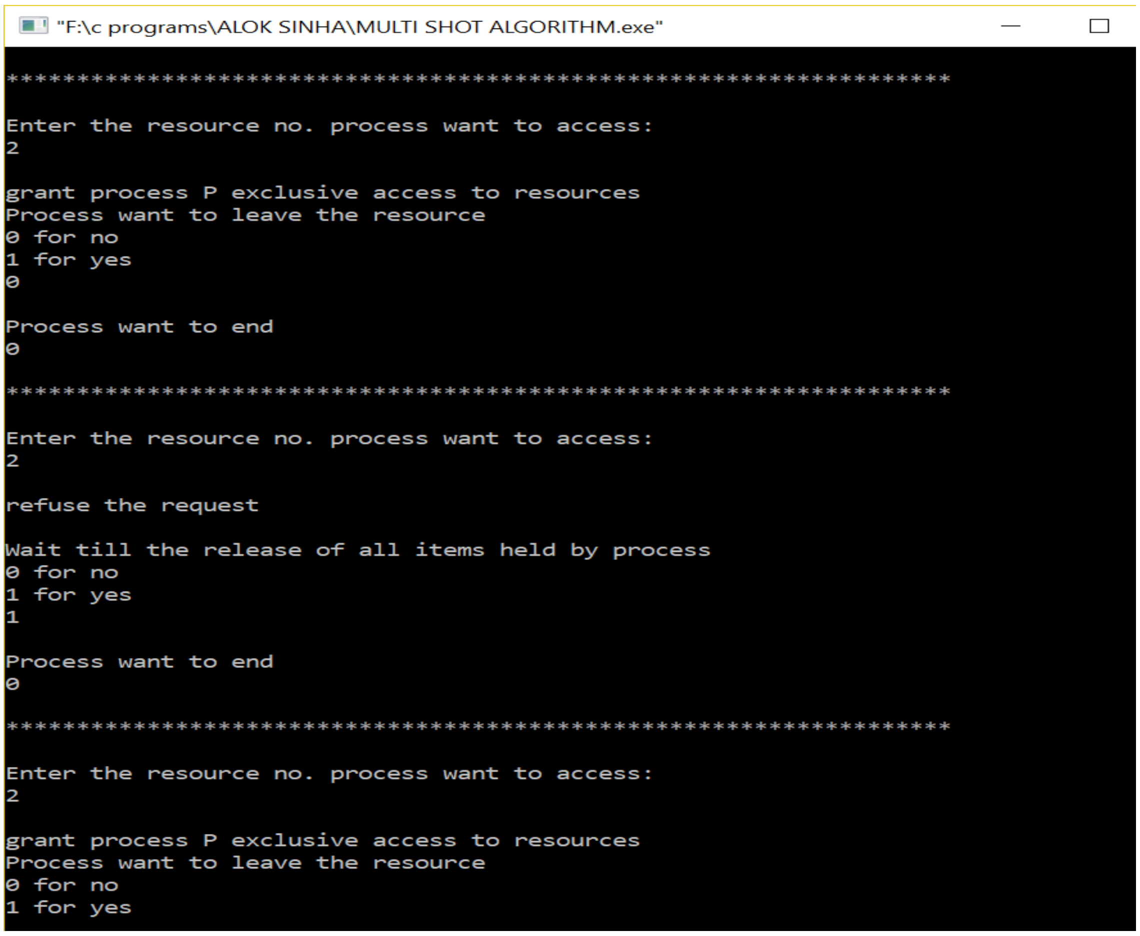
```
#include<iostream>
using namespace std;
int main(){
        int n,m,att=0;
        cout<<"Enter the number of resources:\n";
        cin>>n;
        int res[n];

        cout<<"\nEnter the \n1 for resource NOT FREE \n0 for resource
        FREE:\n"; for(int i=0;i<n;i++){
                cin>>res[i];
        }

        int choice=0;
```

```
            while(choice==0){
            cout<<"\n********************************************************
************\n";
            int count=0,q=0,l=0;
            cout<<"\nEnter the resource no. process want to access:\n";
            cin>>m;
            if(att!=0 ){
            cout<<"\nrefuse the request\n";
            cout<<"\nWait till the release of all items held by process\n0 for no \n1 for
yes\n";
            cin>>l;
            if( l== 1)
    {

    att=0;
        for(int p=0;p<n;p++)
        {
            res[p]=0;
        }
    }
            }
            else if(m>n || m<1)
            cout<<"\nResource does not exists\n";
            else{
                        if(res[m-1]!=0){
                                    cout<<"\nwait until resource free \n";

                                                res[m-1]=0;


                        }
                        else if(res[m-1]==0){
                                    cout<<"\ngrant process P exclusive access to resources\n";
                                    att=1;
                                    res[m-1]=1;


                                    cout<<"Process want to leave the resource \n0 for no\n1 for yes
";
            cin>>q;

            if(q==1)
            {
                att=0;

                res[m-1]=0;
            }
                        }
            }
            cout<<"\nProcess want to end \n";
            cin>>choice;
    }
    return 0;
    }
```



"F:\c programs\ALOK SINHA\MULTI SHOT ALGORITHM.exe"

```
Enter the number of resources:
5

Enter the
1 for resource NOT FREE
0 for resource FREE:
0
0
1
1
0

*******************************************************************

Enter the resource no. process want to access:
3

wait until resource free

Process want to end
0

*******************************************************************

Enter the resource no. process want to access:
2

grant process P exclusive access to resources
Process want to leave the resource
0 for no
1 for yes
0

Process want to end
0

*******************************************************************
```

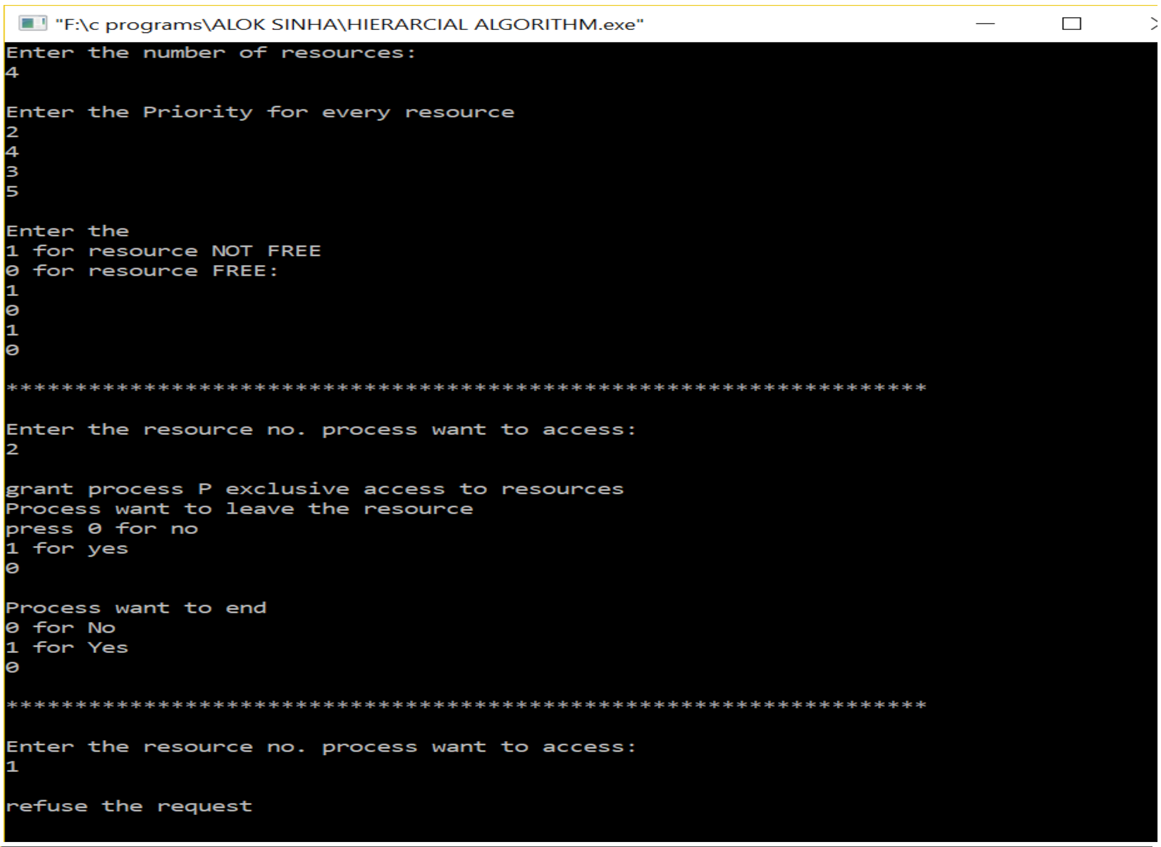


```
*******************************************************************

Enter the resource no. process want to access:
2

grant process P exclusive access to resources
Process want to leave the resource
0 for no
1 for yes
0

Process want to end
0

*******************************************************************

Enter the resource no. process want to access:
2

refuse the request

Wait till the release of all items held by process
0 for no
1 for yes
1

Process want to end
0

*******************************************************************

Enter the resource no. process want to access:
2

grant process P exclusive access to resources
Process want to leave the resource
0 for no
1 for yes
```

HIERARCIAL ALGORITHM

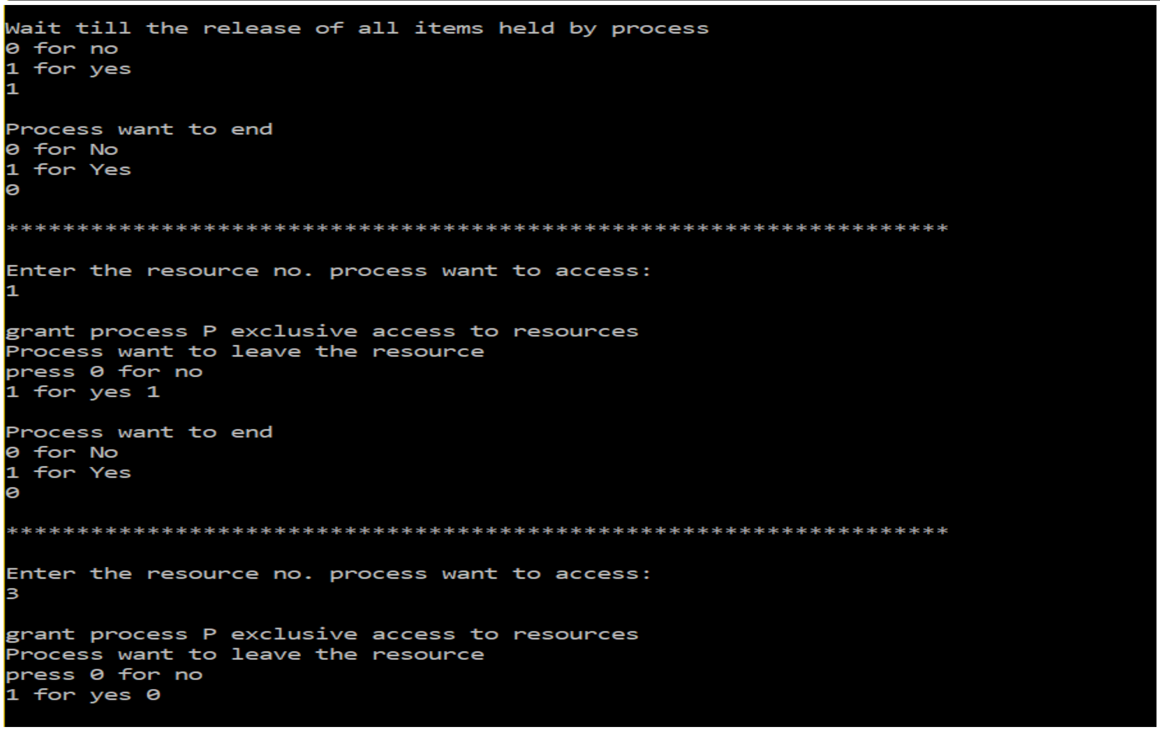
```
#include<iostream>
using namespace std;
int main(){
        int n,m,att=0,pri;
        cout<<"Enter the number of resources:\n";
        cin>>n;
        int res[n],priority[n];

        cout<<"\nEnter the Priority for every resource\n";
        for(int i=0;i<n;i++){
                cin>>priority[i];


        cout<<"\nEnter the \n1 for resource NOT FREE \n0 for resource
        FREE:\n"; for(int i=0;i<n;i++){
                cin>>res[i];
        }

        int choice=0;
```

```
          while(choice==0){
          cout<<"\n**********************************************************
************\n";
          int count=0,q=0,l=0;
          cout<<"\nEnter the resource no. process want to access:\n";
          cin>>m;
          if(att!=0 && pri >= priority[m-1] ){
          cout<<"\nrefuse the request\n";
          cout<<"\nWait till the release of all items held by process \n0 for no \n1 for
yes\n";
          cin>>l;
          if( l== 1)
   {
      att=0;
      pri=0;
      for(int p=0;p<n;p++)
      {
         res[p]=0;
      }
   }
          }
          else if(m>n || m<1)
          cout<<"\nResource does not exists\n";
          else{
                    if(res[m-1]!=0){
                              cout<<"\nwait until resource free \n";

                                        res[m-1]=0;
                                        pri=0;

                    }
                    else if(res[m-1]==0){
                              cout<<"\ngrant process P exclusive access to resources\n";
                              att=1;
                              res[m-1]=1;
                              pri=priority[m-1];

                              cout<<"Process want to leave the resource \npress 0 for no\n1
for yes ";
          cin>>q;
          if(q==1)
          {
             att=0;
             pri=0;
             res[m-1]=0;
          }
                    }
          }
          cout<<"\nProcess want to end\n0 for No\n1 for Yes\n";
          cin>>choice;
}
return 0;
}
```

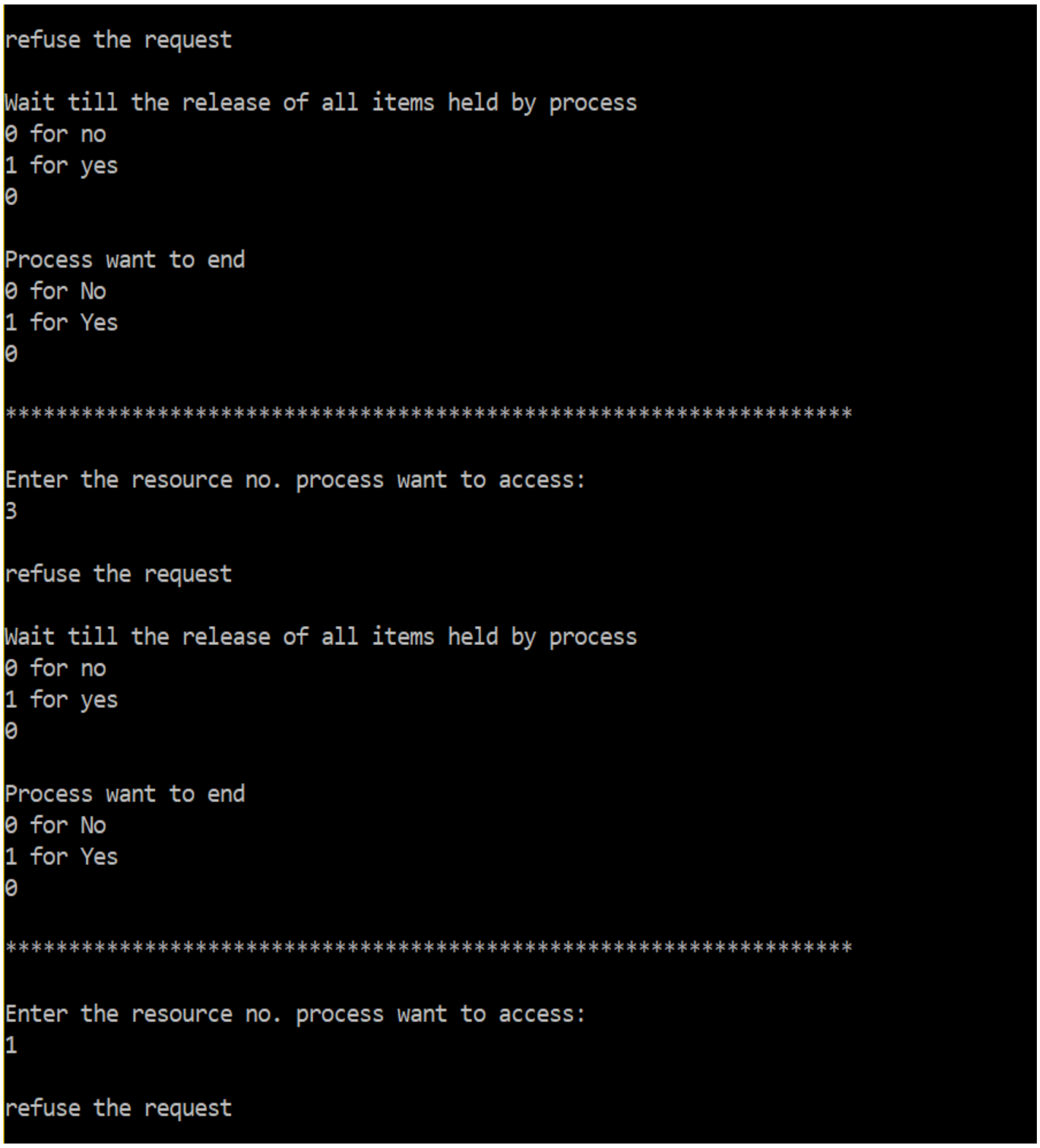


"F:\c programs\ALOK SINHA\HIERARCIAL ALGORITHM.exe"

```
Enter the number of resources:
4

Enter the Priority for every resource
2
4
3
5

Enter the
1 for resource NOT FREE
0 for resource FREE:
1
0
1
0

*******************************************************************

Enter the resource no. process want to access:
2

grant process P exclusive access to resources
Process want to leave the resource
press 0 for no
1 for yes
0

Process want to end
0 for No
1 for Yes
0

*******************************************************************

Enter the resource no. process want to access:
1

refuse the request
```



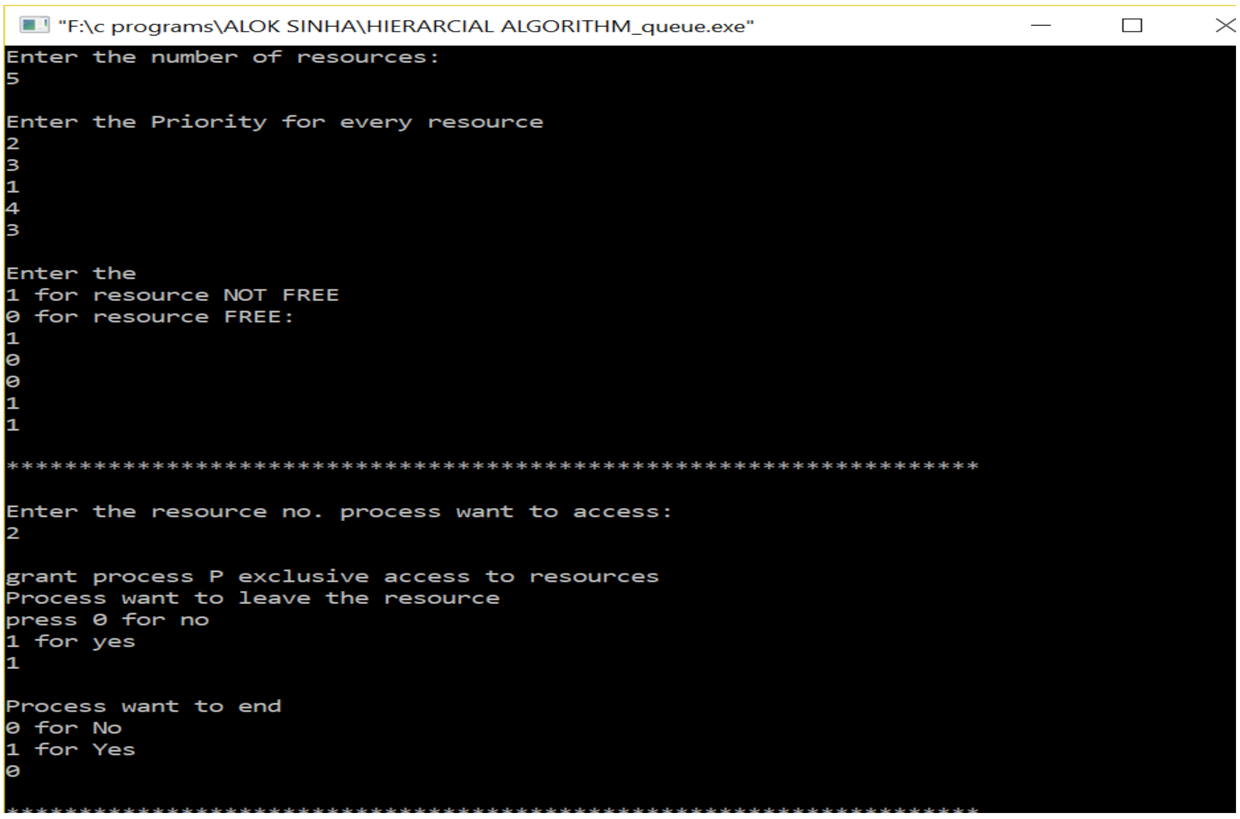
```
Wait till the release of all items held by process
0 for no
1 for yes
1

Process want to end
0 for No
1 for Yes
0

*******************************************************************

Enter the resource no. process want to access:
1

grant process P exclusive access to resources
Process want to leave the resource
press 0 for no
1 for yes 1

Process want to end
0 for No
1 for Yes
0

*******************************************************************

Enter the resource no. process want to access:
3

grant process P exclusive access to resources
Process want to leave the resource
press 0 for no
1 for yes 0
```



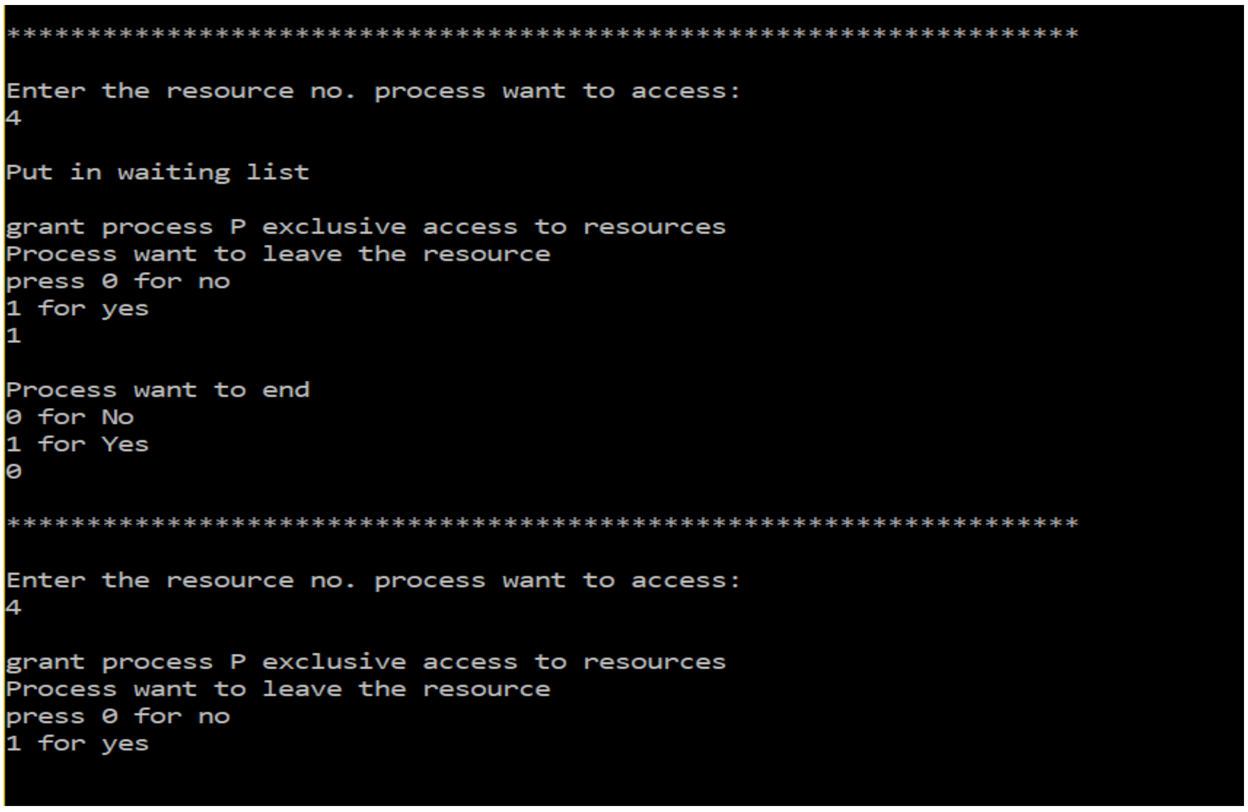
```
refuse the request

Wait till the release of all items held by process
0 for no
1 for yes
0

Process want to end
0 for No
1 for Yes
0

*******************************************************************

Enter the resource no. process want to access:
3

refuse the request

Wait till the release of all items held by process
0 for no
1 for yes
0

Process want to end
0 for No
1 for Yes
0

*******************************************************************

Enter the resource no. process want to access:
1

refuse the request
```

HIERARCIAL WITH QUEUING

```
#include<iostream>

char q[25];
int t=-1;

void push(char a)
{
q[++t]=a;
}

void pop()
{
q[t]=0;
t--;
}

char tos()
{
return q[t];
}

using namespace std;
int main(){
        int n,m,att=0,pri;

        cout<<"Enter the number of resources:\n";
        cin>>n;
        int res[n],priority[n];

        cout<<"\nEnter the Priority for every resource\n";
        for(int i=0;i<n;i++){
                cin>>priority[i];
        }

        cout<<"\nEnter the \n1 for resource NOT FREE \n0 for resource
        FREE:\n"; for(int i=0;i<n;i++){
                cin>>res[i];
        }

        int choice=0;

        while(choice==0){
        cout<<"\n*******************************************************
************\n";
        int count=0,q=0,l=0;
        cout<<"\nEnter the resource no. process want to
        access:\n"; cin>>m;
        if(att!=0 && pri >= priority[m-1] ){
            cout<<"\nrefuse the request\n";
            cout<<"\nWait till the release of all items held by process \n0 for no \n1 for
yes\n";
            cin>>l;
            if( l== 1)
    {
        att=0;
        pri=0;
        for(int p=0;p<n;p++)
        {
            res[p]=0;
        }
    }
            }
            else if(m>n || m<1)
            cout<<"\nResource does not exists\n";
            else{
                        if(res[m-1]!=0){
                                    cout<<"\nPut in waiting list \n";
                                    push('a');
            res[m-1]=0;
            pri=0;

                        }

                        if(res[m-1]==0 || tos()=='a'){
                                    cout<<"\ngrant process P exclusive access to resources\n";
                                    att=1;
                                    res[m-1]=1;
                                    pri=priority[m-1];

                                    cout<<"Process want to leave the resource \npress 0 for no\n1
for yes ";
            cin>>q;

            if(q==1)
            {
                att=0;
                pri=0;
                res[m-1]=0;
            }
                        }
            }
            cout<<"\nProcess want to end\n0 for No\n1 for Yes\n";
            cin>>choice;
}
return 0;
}
```


"F:\c programs\ALOK SINHA\HIERARCIAL ALGORITHM_queue.exe"

```
Enter the number of resources:
5

Enter the Priority for every resource
2
3
1
4
3

Enter the
1 for resource NOT FREE
0 for resource FREE:
1
0
0
1
1

*******************************************************************

Enter the resource no. process want to access:
2

grant process P exclusive access to resources
Process want to leave the resource
press 0 for no
1 for yes
1

Process want to end
0 for No
1 for Yes
0

*******************************************************************
```



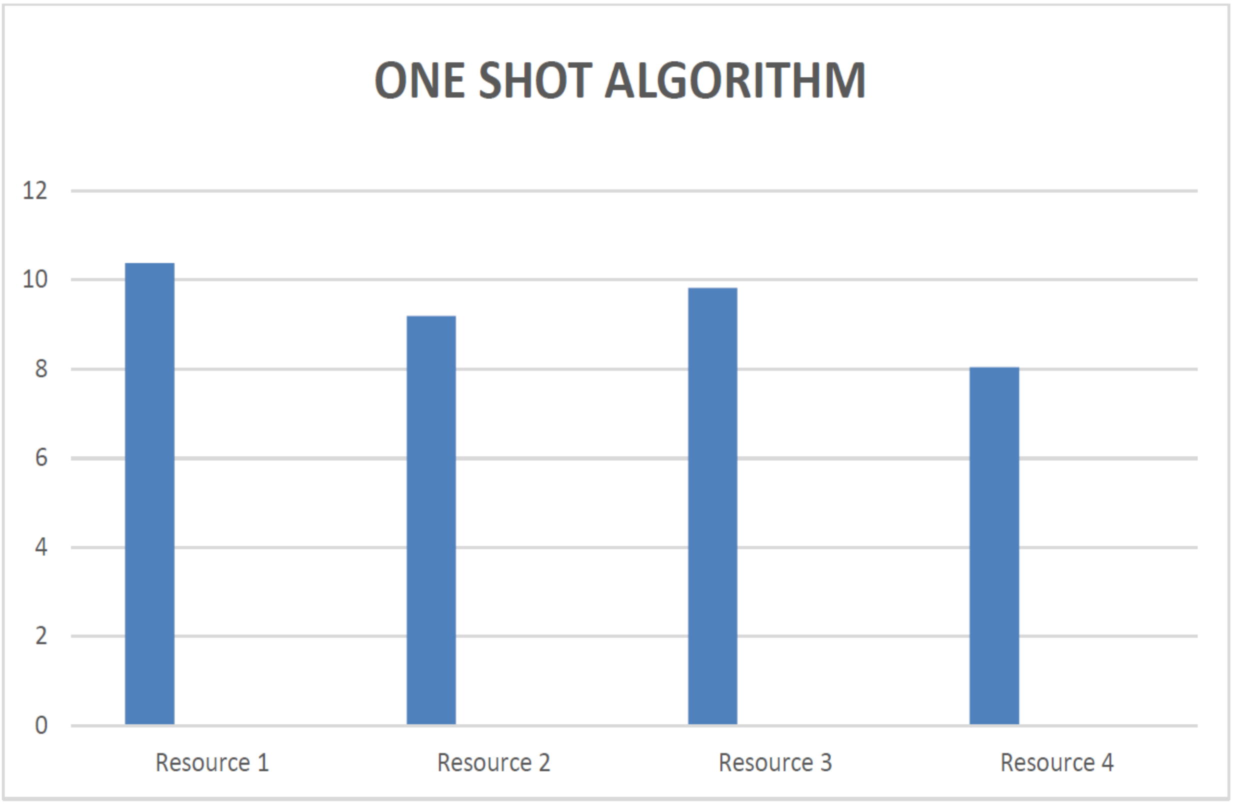
```
*******************************************************************

Enter the resource no. process want to access:
4

Put in waiting list

grant process P exclusive access to resources
Process want to leave the resource
press 0 for no
1 for yes
1

Process want to end
0 for No
1 for Yes
0

*******************************************************************

Enter the resource no. process want to access:
4

grant process P exclusive access to resources
Process want to leave the resource
press 0 for no
1 for yes
```

ANALYSIS:

ONE SHOT ALGORITHM	**MULTI SHOT ALGORITHM**	**HIERARCHAL ALGORITHM**	**HIERARCHAL WITH QUEUING ALGORITHM**
A process can grant a single resource at a time.	A process can grant multiple instances of resource but it should hold only one instance at a time.	A process can grant multiple resource with a priority index.	A process can grant multiple resource with the priority and arranging the resources in a order of queue according to their priority.
In one shot algorithm, a process can’t grant the resource if also there is free resource available after allocating a resource.	In multi shot algorithm, it refuses the request when a process is holding any resource.	In hierarchal algorithm, it refuses the request if demanding process has less priority than the process which is using the resource.	In hierarchal queueing algorithm, it ensures that the waiting done by the processes is done on a series of queues as per resource same as hierarchal but here we use waiting of the process in a queue.
This application can be used in some type of game or the online contest in which user can take part only one time.	This application can be implemented in some real time applications like dining philosopher where a person request for a resource(knife) but it can be only accessible if and only if the resource is free.	The application can be implemented in some operating system like in scheduling or process occur with the priority.	This application can also be implemented in operating system where multiple process access in priority but it assures the process must be in queue.
In terms of usability, this algorithm doesn’t work because even the free resources after allocation it can’t be allocated.	In terms of usability, this algorithm is far better in terms of requesting multiple resources but it can be accessible to only one resource at a time.	In terms of usability, the priority has assigned for the process to assign the resource so that the highest priority get chance first to perform action.	In terms of usability, the priority with queue is assigned for the resources as the process request it gets in queue by the priority sequence.

Suppose that a person needs both a knife and fork to eat. Person P1 needs knife and fork and person P2 needs knife and fork. person P1 requests knife and fork, person P2 also requests knife and fork. Only one of P1 or P2 will be granted the resources. Suppose it is P1. P2 is forced to wait. Person P1 uses the knife and fork until finished. Person P1 releases the knife and fork. Since the resources that P2 was waiting for are free, P2 is granted both the knife and fork. Person P2 uses the knife and fork until finished. Person P2 releases the knife and fork.	If a process P wants to request resources while holding resources, they follow these steps: P frees all resources being held P requests all resources previously held plus the new resources it wants to acquire. Like if person request for the knife and the person holding fork then request refuses. And the person has to free knife and again it has to request for the knife or fork.	Suppose that a person needs both a knife and fork to eat. Knife is given priority 2 and fork priority 1, assuming that zero is the highest priority. Person P1 needs knife and fork and person P2 needs knife and fork. Person P1 requests knife first because it is lower priority and then fork. Person P2 also requests knife and then fork. Only one of P1 or P2 will be granted the knife. Suppose it is P1. Person P2 will wait for the knife to be free. Then only P1 will request the fork. The request will be granted. Person P1 will use the knife and fork until finished. Person P1 will release the knife and fork. Since person P2 is waiting for the knife, the request can now be granted. Then Person P2 will immediately request the fork and this request will also be granted. Person P2 will use the knife and fork until finished. Person P2 will release the knife and fork.	Suppose that a person needs both a knife and fork to eat. Knife is given priority 2 and fork priority 1, assuming that zero is the highest priority. Person P1 needs knife and fork and person P2 needs knife and fork. Person P1 requests knife first because it is lower priority and then fork. Person P2 also requests knife and then fork. Only one of P1 or P2 will be granted the knife. Suppose it is P1. Person P2 will be put in the queue for the knife. Then only P1 will request the fork. The request will be granted. Person P1 will use the knife and fork until finished. Person P1 will release the knife and fork. Since person P2 is at the front of the queue waiting for the knife; person P2 will be granted the knife. Then Person P2 will immediately request the fork and this request will be granted. Person P2 will use the knife and fork until finished. Person P2 will release the knife and fork.

<u>ONE SHOT ALGORITHM</u>



INDEX

X- AXIS = NO. OF RESOURCES

Y-AXIS = TIME TAKEN FOR EXECUTION OF THE PROCESS (IN SECOND)

For the process holding resource 1, it took 10.691s for the execution.

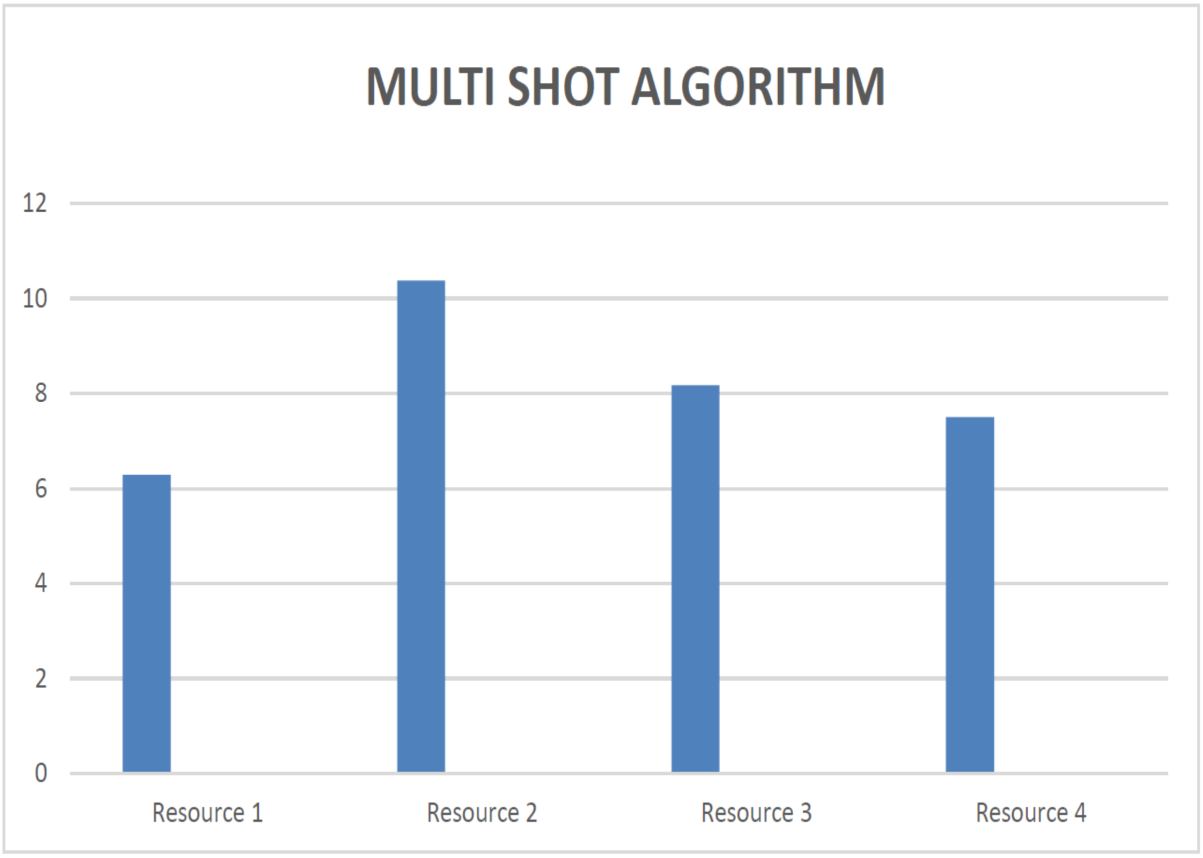
For the process holding resource 2, it took 9.190s for the execution.

For the process holding resource 3, it took 9.812s for the execution.

For the process holding resource 4, it took 8.039s for the execution.

Average execution time for the taken resources = 9.432s

MULTI SHOT ALGORITHM



INDEX

X- AXIS = NO. OF RESOURCES

Y-AXIS = TIME TAKEN FOR EXECUTION OF THE PROCESS (IN SECOND)

For the process holding resource 1, it took 6.283s for the execution.

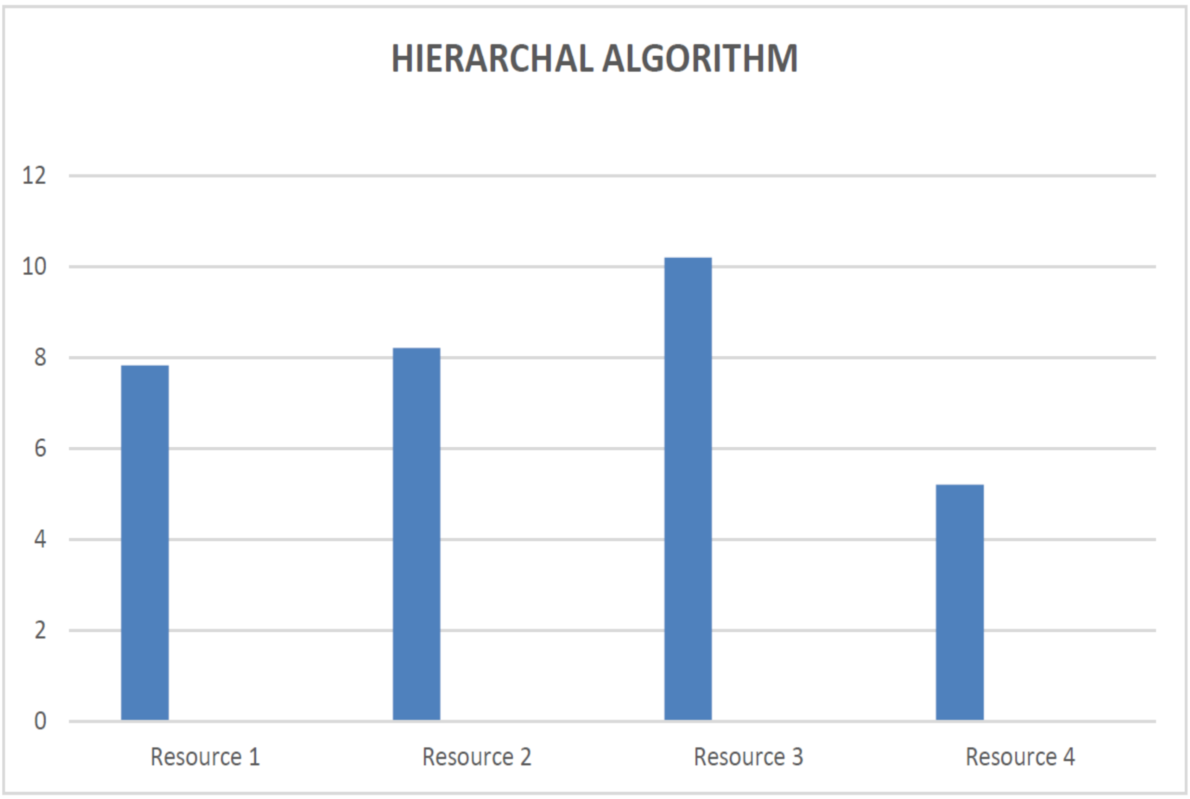
For the process holding resource 2, it took 10.373s for the execution.

For the process holding resource 3, it took 9.973s for the execution.

For the process holding resource 4, it took 7.50s for the execution.

Average execution time for the taken resources = 8.259s

HIERARCHAL ALGORITHM



INDEX

X- AXIS = NO. OF RESOURCES

Y-AXIS = TIME TAKEN FOR EXECUTION OF THE PROCESS (IN SECOND)

For the process holding resource 1, it took 7.82 for the execution.

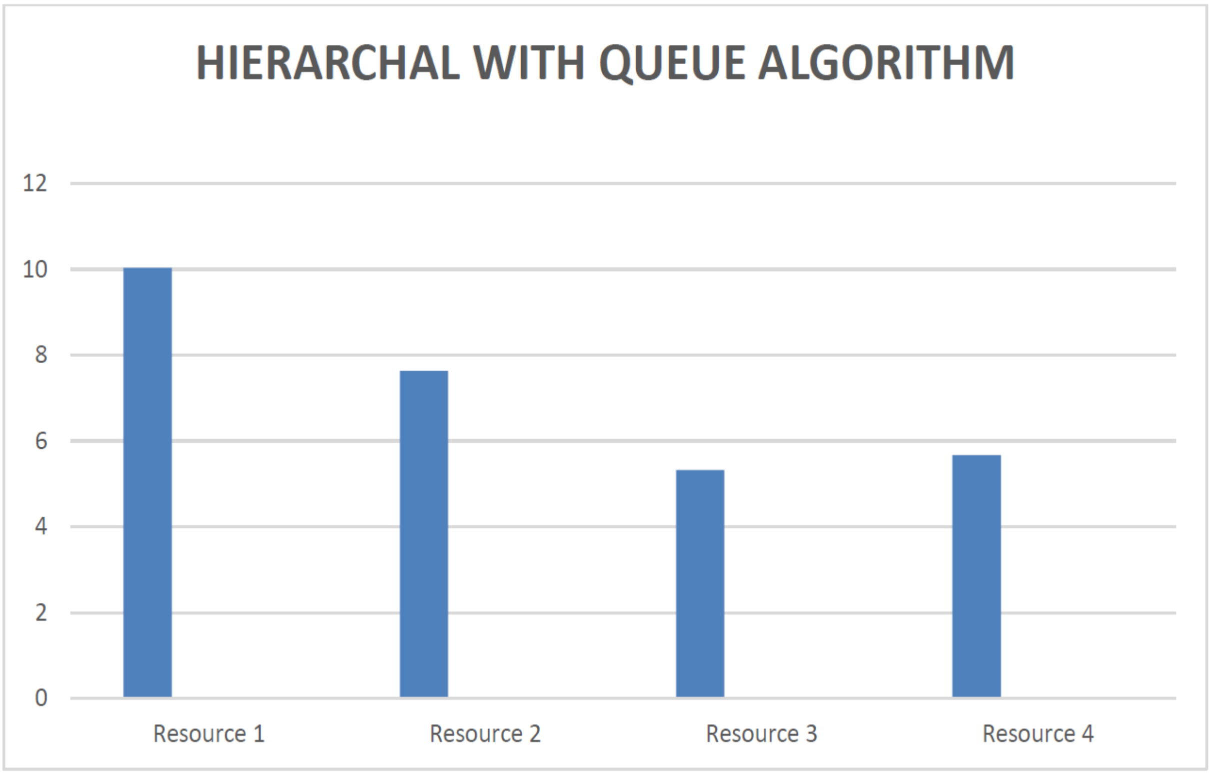
For the process holding resource 2, it took 8.212s for the execution.

For the process holding resource 3, it took 10.204s for the execution.

For the process holding resource 4, it took 5.203s for the execution.

Average execution time for the taken resources = 7.860s

HIERARCHAL WITH QUEUE ALGORITHM



INDEX

X- AXIS = NO. OF RESOURCES

Y-AXIS = TIME TAKEN FOR EXECUTION OF THE PROCESS (IN SECOND)

For the process holding resource 1, it took 10.03s for the execution.

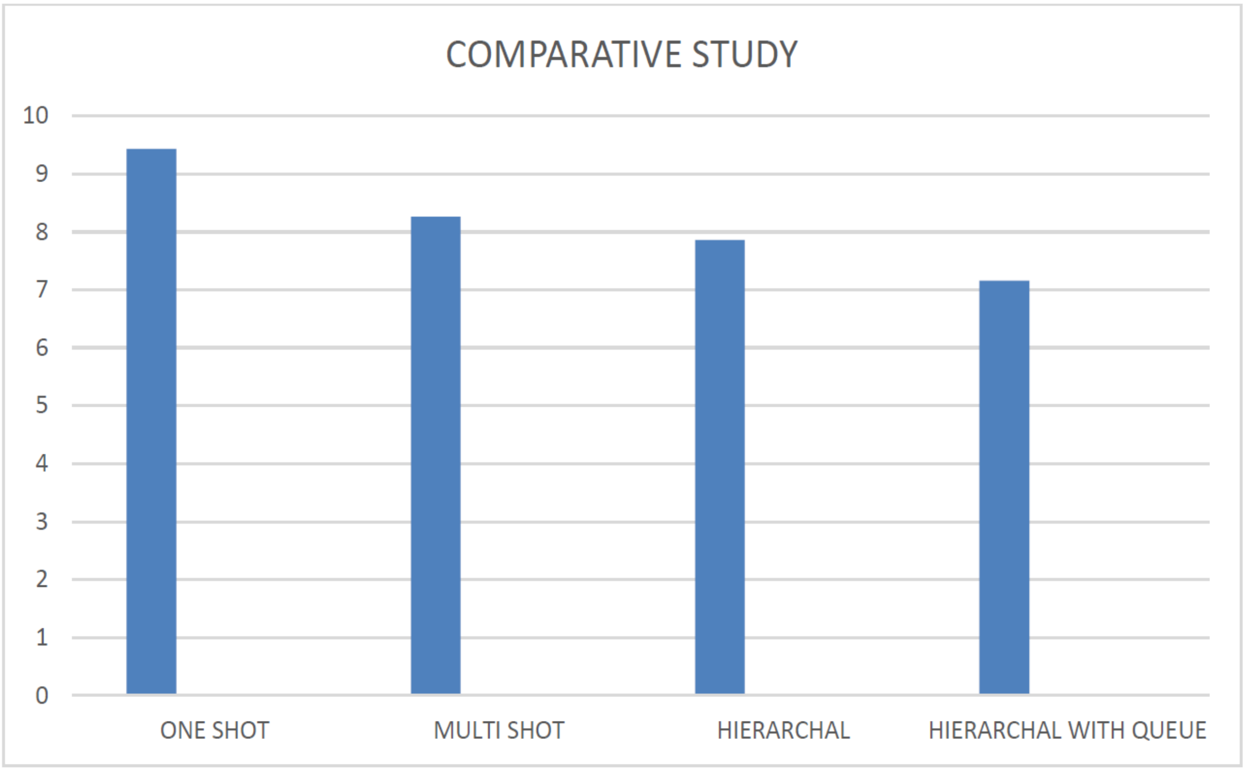
For the process holding resource 2, it took 7.627s for the execution.

For the process holding resource 3, it took 5.32s for the execution.

For the process holding resource 4, it took 5.67s for the execution.

Average execution time for the taken resources = 7.167s

COMPARATIVE STUDY



INDEX

X – AXIS: ALGORITHMS

Y – AXIS: AVERAGE EXECUTION TIME OF RESPECTIVE ALGORITHMS.

Average execution time of One-Shot Algorithm = 9.432s

Average execution time of Multi Shot Algorithm = 8.259s

Average execution time of Hierarchal Algorithm = 7.86s

Average execution time of Hierarchal with queue Algorithm = 7.16s

From the above bar graph, we conclude that the execution time of the hierarchal with queue is far better than the others algorithm with respect to the utilization of resources. A process can grant multiple resource with the priority and arranging the resources in a order of queue according to their priority. So hierarchal is better than other in terms of the execution time as well as it prevents from the deadlock condition.

ONE SHOT ALGORITHM:

On running one shot algorithm for many different cases, we conclude that processes cannot access any resource after it has used some resource. So in most of the cases when the request is refused. Even if the resource is free process cannot use it .So this algorithm is not efficient and also should be used only for some specific applications like some type of game or the contest in which user can take part only one time.

MULTISHOT ALGORITHM:

Multi shot algorithm is way better than one shot algorithm because it end one of the necessary condition for deadlock that is hold and wait. It refuse the request only if the process is holding any resource. Also it gives the permission to use the resource if it is free. So overall it is better algorithm and can be used in real time application.

HIERARCHICAL ALGORITHM:

Hierarchical algorithm is like multishot algorithm but the only difference is that it assigns priority to various processes. It only refuses the request if demanding process has less priority than the process which is using the resource. One of the benefit it provides is that it can be used in system where system processes need to be executed before other processes. So hierarchical algorithm is more suitable for operating system.

HIERARCHICAL ALGORITHM (WITH QUEUE)

An interesting variation on the Hierarchical Algorithm can be created by ensuring that the waiting done by the processes is done on a series of queues, one per resource. This is same algorithm as the hierarchical algorithm but the only difference is we use a queue for the waiting of processes. When process come to front of queue it will be given the resource. So hierarchical algorithm with queue is suitable for operating system.

CONCLUSION

By implementing above mentioned algorithm we can easily detect the condition of deadlock and can successfully handle the situation of deadlock.

REFERENCES:

[1]. Luca Padovani,"TypeState-Oriented Programming(TSOP) Deadlock- free oriented programming",http://doi.org/10.22152/programming-journal.org%2F2018%2F2%2F15, 2018

[2]. Zana Ghaderi, Ayed Alqahatani, Nader Bagherzadeh,"aging aware deadlock free adaptive routing algorithm and online monitoring in 3D",http://doi.org/10.1109/TPDS.2017.2780173,2018

[3]. Wiktor B. Daszczuk,"Deadlock Detection in Distributed System"2017

[4]. M.D.Gan, Z.J.Ding,"Deadlock control of multithread software",http://doi.org/10.1109/ICNSC.2016.7478975,2016

[5]. Vandana Kate,Akansha Jaiswal, Ambika Gehlot,"Survey on distributed deadlock and distributed algorithms to detect and resolve deadlock.

[6]. Yan Cai, k.Zhai, Shngru wu, W.k. Chan," Synchronizing Threads Globally Detect Real Deadlocks For Multithreaded Programs", ACM 2013

[7]. Jeremy D.Buhler, Kunal Agrawal, Peng Li, Roger D. Chamberlain, "Efficient Deadlock Avoidance For Streaming Computation With Filtering", 2012 ACM, February 2012.

[8]. Kunwar Singh Vaisla, Menka Goswami, Ajit Singh, "VGS Algorithm - an Efficient Deadlock Resolution Method

[9] Predicting the Time to Migrate Into Deadlock Using a Discrete Time Markov Chain: https://dl.acm.org/citation.cfm?doid=3185768.3186403

[10] Petri Nets and Deadlock-Free Scheduling of Open Shop Manufacturing Systems https://ieeexplore.ieee.org/document/7946179

[11] Deadlock-Free Typestate-Oriented Programming: http://programming-journal.org/2018/2/15

[12] Graceful deadlock-free fault-tolerant routing algorithm for 3D Network-on-Chip architectures: https://www.sciencedirect.com/science/article/pii/S0743731514000045?via%3Dihub

[13] Deadlock Property Analysis of Concurrent Programs Based on Petri Net Structure: https://link.springer.com/article/10.1007%2Fs10766-016-0440-7

[14] Deadlock control of multithreaded software based on Petri nets: A brief review: https://ieeexplore.ieee.org/document/7478975

[15] Congestion Aware Routing for On-Chip Communication in NoC Systems: https://link.springer.com/chapter/10.1007%2F978-3-319-61566-0_50

[16] Predicting the Time to Migrate Into Deadlock Using a Discrete Time Markov Chain: https://dl.acm.org/citation.cfm?doid=3185768.3186403

[17] Petri Nets and Deadlock-Free Scheduling of Open Shop Manufacturing Systems: https://ieeexplore.ieee.org/document/7946179

[18] High-Performance, Low-Complexity Deadlock Avoidance for Arbitrary Topologies/Routings : https://dl.acm.org/citation.cfm?doid=3205289.3205307

[19] Robust Deadlock Avoidance for Sequential Resource Allocation Systems With Resource Outages : https://ieeexplore.ieee.org/document/7997993

[20] Deadlock control of multithreaded software based on Petri nets: A brief review: https://ieeexplore.ieee.org/document/7478975

[21] http://www2.cs.uregina.ca/~hamilton/courses/330/notes/synchro/deadlock-ex.html

ROLE AND RESPONSIBILITIES OF EACH TEAM MEMBERS.

ALOK SINHA – CODING

MANISH PAIKARA – CODING

BIBEK SINGH – TOOL RESEARCH

SUMIT KUMAR SAH – RESOURCE MANAGEMENT

RAHIL RAIKWAR – RESEARCH