PROJECT REPORT

CSE- 2005- OPERATING SYSTEM

DEADLOCK PREVENTION TECHNIQUES

Group 10

Submitted To-
Dr. Vijay Rajan V

School of Computer Science and Engineering

DECLARATION

I/We hereby declare that the project entitled **"Deadlock Prevention"** submitted by us to the School of Computer Science and Engineering, VIT University, Vellore-632014 in partial fulfillment of the requirements for the project of **Operating System** is a record of bonafide work carried out by me/us under the supervision of **Dr Vijay Rajan V,** I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma of this institute or of any other institute or university.

Signature

Apoorv Raizada(15BCE2072)

Signature

Arpit Khurana(15BCE0353)

Signature

Chirag Arora(15BCE0532)

Signature

Shreyansh Jain(15BCE0873)

School of Computer Science and Engineering

CERTIFICATE

The project report entitled "**DEADLOCK PREVENTION**" is prepared and submitted by **Candidates Apoorv Raizada (15BCE2072), Arpit Khurana (15BCE0353) ,Chirag Arora (15BCE0532) and Shreyansh Jain(15BCE0873).** It has been found satisfactory in terms of scope, quality and presentation as partial fulfillment of the requirements for the project of **Operating System** in VIT University, India.

Guide
(Name & Signature)

Internal Examiner
(Name & Signature)

External Examiner
(Name & Signature)

ACKNOWLEDGEMENT

It is our privilege to express our sincerest regards to our project coordinator, Dr. Vijayarajan V, for their valuable inputs, able guidance, encouragement, whole-hearted cooperation and constructive criticism throughout the duration of our project.

I deeply express our sincere thanks to our Head of Department and Dean for encouraging and allowing us to present the project on the topic "Deadlock prevention algorithms" for operating system course.

I take this opportunity to thank all our lecturers who have directly or indirectly helped our project. We pay our respects and love to our parents and all other family members and friends for their love and encouragement throughout our career. Last but not the least we express our thanks to our friends for their cooperation and support.

Signature

Apoorv Raizada(15BCE2072)
Arpit Khurana(15BCE0353)
Chirag Arora(15BCE0532)
Shreyansh Jain(15BCE0873)

CONTENTS

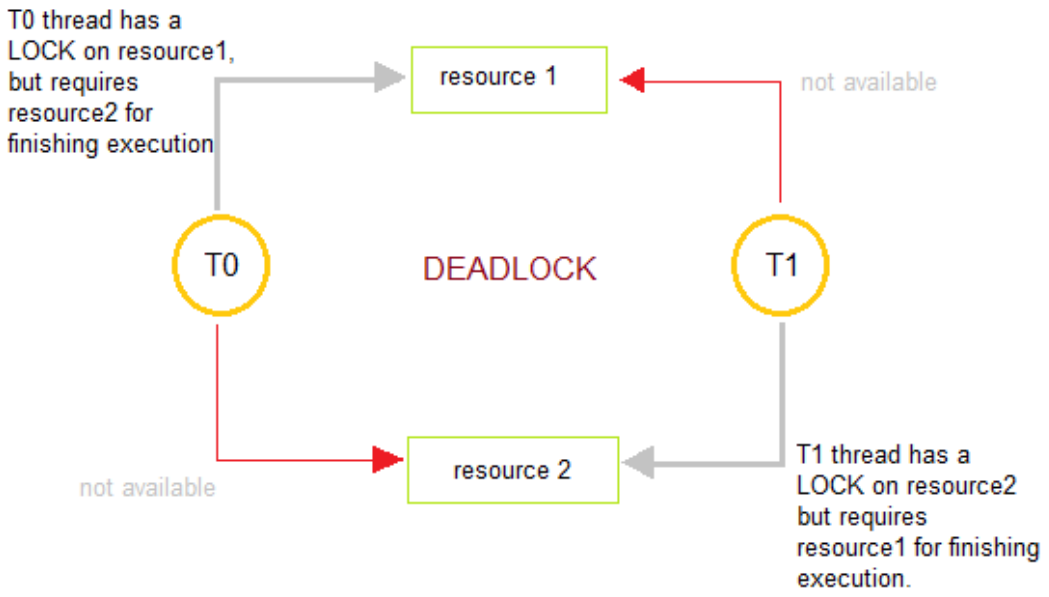
Abstract:

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.



CONDITIONS FOR DEADLOCK

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:

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.

LITERATURE SURVEY+ METHODOLGY

One-shot Algorithm

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 if

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 Ps releases the knife and fork.

MULTISHOT ALGORITHM

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

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)

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

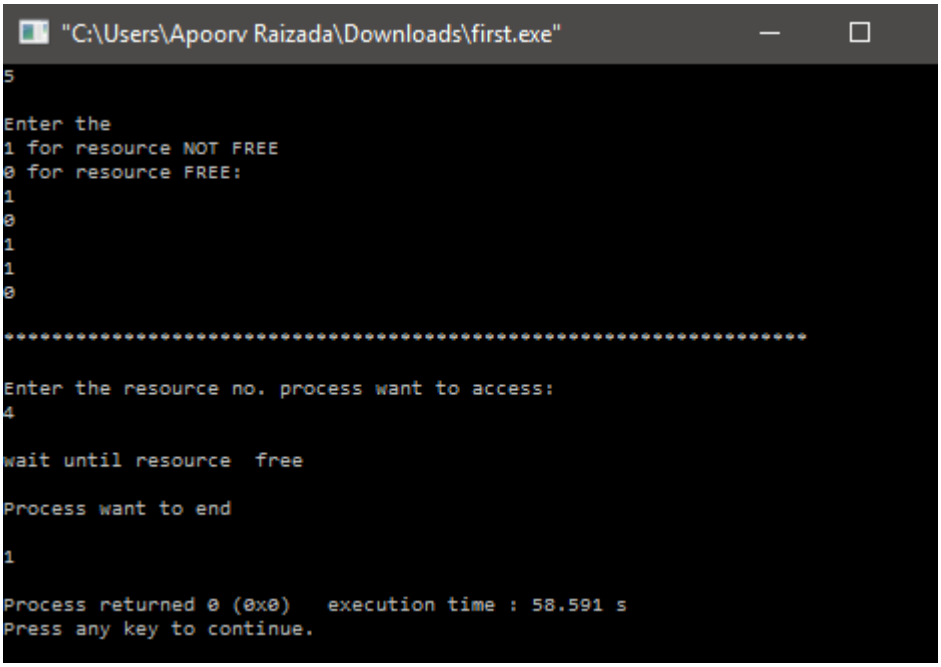
i) We will enter all the resources required by the processes.
ii) The request will automatically be rejected if the demand of resources is greater than the no of resources present.
iii) Then we will find if safe sequence is possible or not.
iv) Then we will implement all the algorithm, and find the arrangement 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(){
```

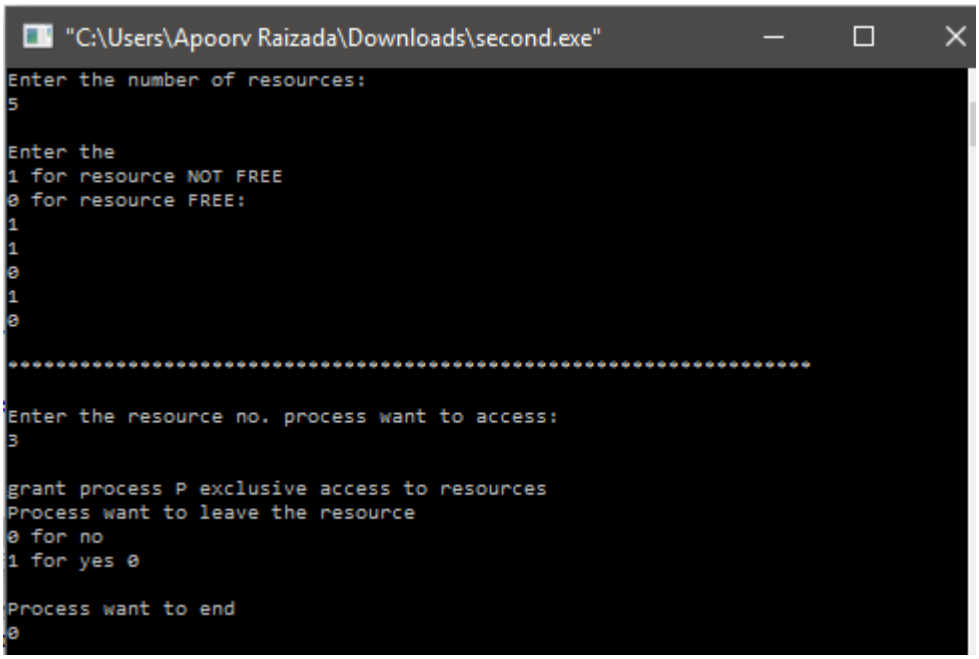
```
        int n,m,att=0;
        cout<<"Enter the number of resources:\n";
        cin>>n;
        int res[n];

        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;
}
```

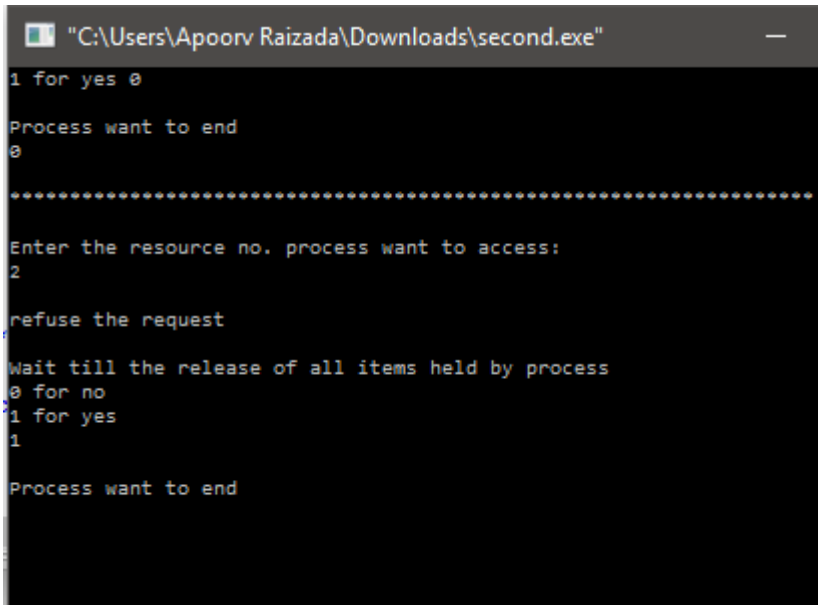


MULTISHOT ALGORITHM

```
#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;
}
```



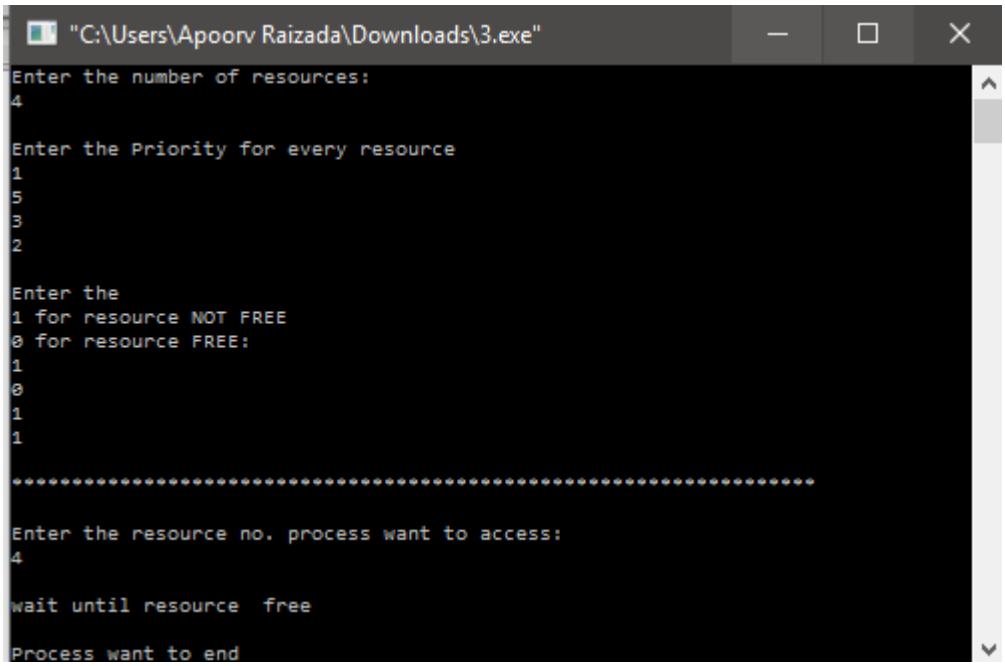


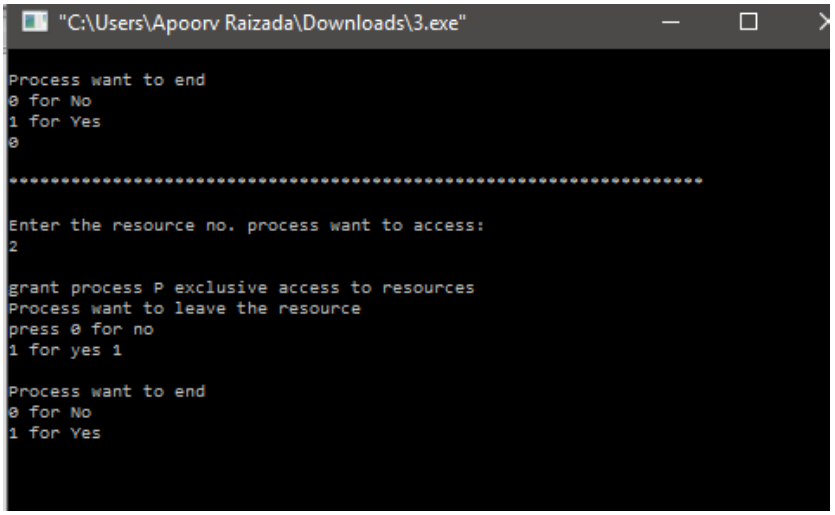
Hierarcial Algorithm

```
#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;
}
```

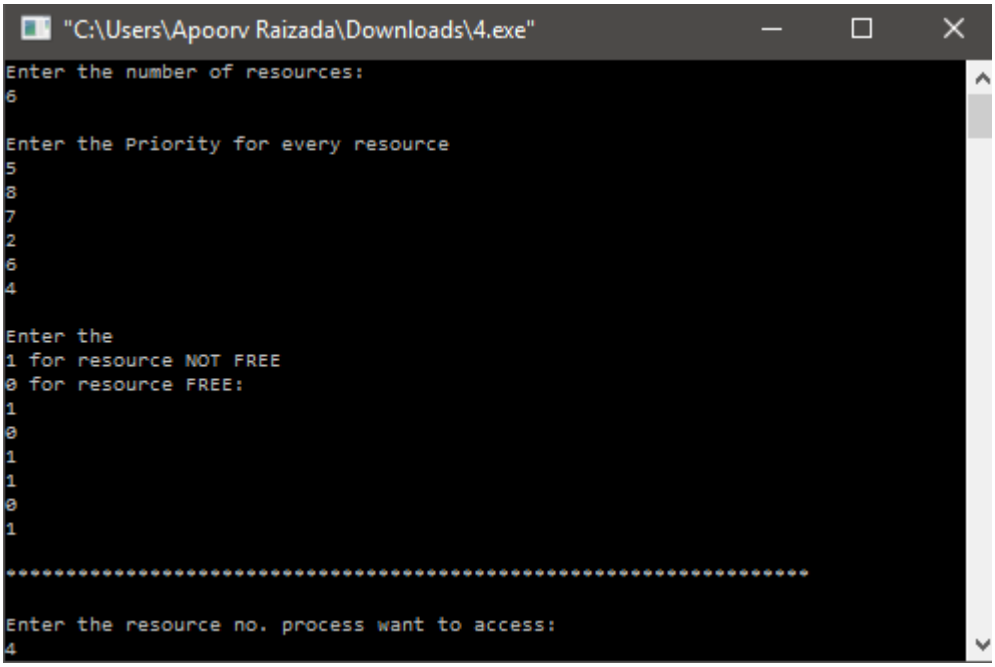


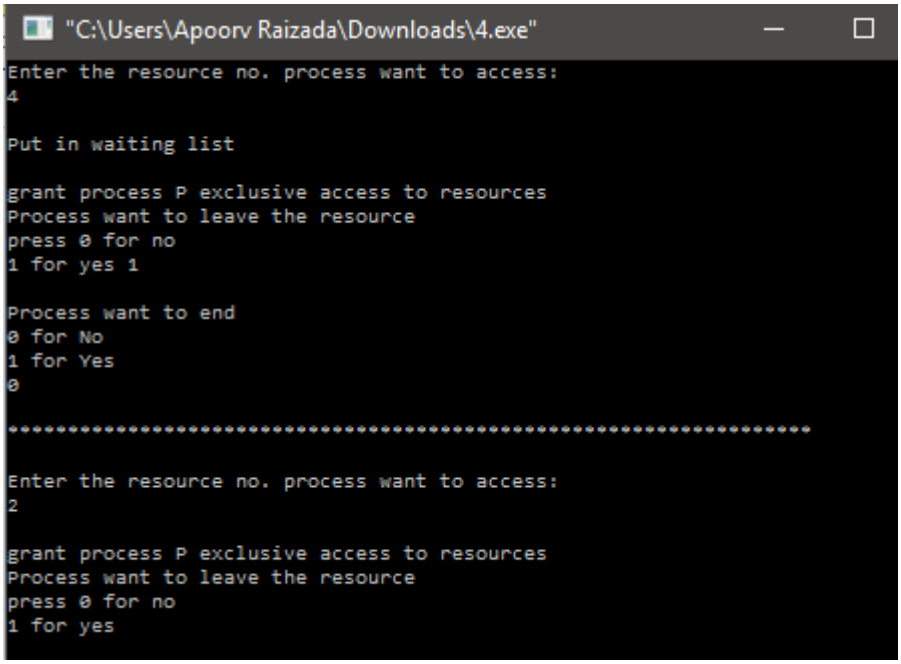


Hierarcial With Queing

```
#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;
}
```





Analysis:

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 resquest 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 an 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.

CONCLUSIONS

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

References:

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

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

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

[4]. Selvaraj Srinivasan, R. Rajaram, "A decenreralized deadlock detection and resolution algorithm for generalized model in distributed systems", january 2011.

[5]. Selvaraj Srinivasan, R. Rajaram, "A decenreralized deadlock detection and resolution algorithm for generalized model in distributed systems", January 2011.

[6]. Iryna Felko, TU Dortmund, “Simulation based Deadlock Avoidance and Optimization in Bidirectional AGVS”, 2011 ACM, march 2011.

[7]. Prodromos Gerakios, Nikolaso Papaspyrou, Konstantinos, Vekris,”Dynamic Deadlock Avoidance in System Code Using Statically Inferred Effects”, 2011 ACM, October 2011.

Peng Li, kunal agrawal, JeremyBuhler, Roger D. Chamberlain,”Deadlock Avoidance for Streaming Computations with Filtering”, 2010 ACM, June 2010