Project Report

**A DYNAMIC AND IMPROVED IMPLEMENTATION OF BANKERS ALGORITHM**

Operating Systems Project

Submitted by

**17BCI0104 – S.Vivekananda Varma**

**Guidance of**

**Prof. Manikandan K**

Abstract:-

Banker’s algorithm can described as resource allocation and deadlock avoidance algorithm which make sure the execution safety by proceeding the allocation of already determined maximum possible of resources and makes the system into the safe state by making a check for all the possible deadlock conditions for all other pending resources. It must know how many resources can a process request. Number of processes is static in general banker’s algorithm, but in most of the system processes vary dynamically and no additional process will start while it is in execution. General bankers algorithm is said to be too conservative in industry as future resources are well-defined by present systems by part-routes. In this project an approach for Dynamic banker’s algorithm is proposed and will be implemented. The algorithm performs the process arrangement on the basis of their need resources that leads to solve the processes in less time. The resources for future processes are included when a process in run time enters safe state. But the before processes are given preference over future process. This prevents the system to fall in unsafe state by allowing the number of resources to be changed at run time.

Introduction:-

If each process in the queue is waiting for a resource that only another process in the queue can cause, then deadlocks happen. This event occurs due to currently held resource. No process execution happens and resources are not released for others. A deadlock occurs when a process enters into the waiting state because requested resources are held by another process in waiting, which already has been waiting for an another resource which is already holded by some other waiting process.

If following four conditions occurs simultaneously then there is a possibility for occurring an deadlock.

1. Mutual exclusion: At a time, only a single process can be executed and non sharable resource is utilized.
2. Hold and wait : Additional resources can be requested if process has atleast one resource.
3. No pre-emption : Only process can release resources voluntarily which it is holding. So, previously had resources should not be taken frcebily from process.
4. Circular wait : If let us take n processes and p1 requested resources are holded by p2, p2 resources are holded by p3, and so on and at last pn requested resource holded by p1.

Literature Survey:-

1) Shiv Prasad, Amiya Dhar Dwivedi, Pankaj Kawadkar, in their research work "Deadlock Avoidance based on Banker’s Algorithm for Waiting State Processes" proposed algorithm for deadlock avoidance technique used for Waiting State processes.

Improvements made in their work:

It says that if processes are going to the waiting state then the consideration of number of allocated resources as well as need of resources in order to select a waiting process for the executing state will make the Banker’s algorithm more efficient than before.

Limitation:-

This paper work did not give any solution when the system is in unsafe state.

2) C.Srinivasulu, Smriti Agrawal and Madhavi Devi in their research work “A Total Need based Resource Reservation Technique for Effective Resource Management” proposed a method that suggests about reserving some resources that ensure at least one process which can be completed after it.

Improvements made in their work:

The research result indicates that frequency of deadlocks has been reduced by approximation of 75% for higher load as compared to the Deadlock Recovery technique, while it tends to be zero for lower load.

Limitation:-

It was unable to provide the details for processes and resources that causes the deadlocks if executed or no safe sequence.

3) Sheau-Dong Lang in his research work “An Extended Banker’s Algorithm for Deadlock Avoidance” proposed an approach for the safety sake in banker’s algorithm.

Improvements made in their work:

Based on resource-related calls of control flow computes the maximum resources claims with prior to the process execution.At runtime, this information is used to basically test the system safety using the General banker’s algorithm.

Limitation:-

But this approach was unable to recognize patterns of resourcerelated calls in real-time system and the practicality was also not proven.

# Software and Hardware:-

There is no Hardware component used in this project.

Software used :- CODE BLOCKS(“C “programming)

Existing Approach:-

You are given no of resources and number of processes among which they have to be allocated so that all processes are taken to safe state. Existing approach called bankers algorithm gives a way for getting safe sequence of processes by avoiding deadlock problem. The allocated resources, maximum resources for each process are given. Also available resources are also given. With that of maximum and allocated for each process get the need resources for each resources with difference of maximum and allocated resources. Now all the processes are checked for their execution by comparing their each need with available. If available greater than or equal to need of any process then process is executed and it releases it maximum resources and they are added to available. This repetition takes place until all processes are in safe state.

a. Available[m] indicates resources available for each resource.

b. Max[n][m] indicates the maximum resources for each process.

c. Allocation[n][m] indicates the allocated resources fro each process

d. Need[n][m] indicates the neede resources for execution for each process.

Note:- Need[m][n] = Max[m][n] - Allocation[m][n]

Limitations of Banker’s Algorithm :-

1. The resources a process could possibly request it must know at first.
2. Here the number of processes are static but many systems it is dynamically there.
3. Here, the number of resources when a process terminated is sufficient for algorithm execution but it is not in general systems.
4. Waiting for resources for hours(many days) is not actually possible.
5. when system is not in safe state it does not give any information. When a process is failed during the execution for it there is no information and also why is it entered unsafe state reason information is not given.
6. If many processes are given at a time then bankers algorithm takes a lot of time for which system cant wait that much time for execution.

Proposed approach:-

Proposed approach allocates the resource automatically to the stopped process for the execution and will always give the appropriate safe sequence for the given processes. Whenever a process is calculated for resources if need is less then directly allocate the resources but if the need resources for the process is more than available resource then the process is sent into a stack. Here in stack whenever processes enter the need arrangement will be done and minimum need will be on the top always. So for a set of processes when once check of need is completed then directly the processes in stack are executed. This decreases the time required for process execution. So this proposed concept is time efficient then original banker algorithm. And also when future resources come they follow first in first execute algorithm. That is the processes already in stack will be executed first and after completion of stack processes future resource is executed. So we can say that the future process which come will be added at bottom of stack directly and wait for execution. This decreases the conservative nature of the original banker’s algorithm.

The average execution time will be increased for both the original and proposed techniques as the processes increases. More processes will lead to higher contention to the resources and more frequently occuring deadlocks. The performance of the proposed approach is better for big range because the overhead which is involved in the resource allocation will be much lower than that of the original banker’s algorithm. The average improvement is approximately 9% over original bankers algorithm.

Algorithm:-

1. FOR Process (i)

WHERE i=0 to n-1

CHECK Need (i)

1. IF Need [i][j]<= Available[i]

THEN

a. Available[i] = Available[i] – Need [i][j];

b. execute [i];

c. Available = Available + Allocation;

d. process execute

e. next process starts

3. ELSE

insert need into stack;

4. sort the stack;

5. end.

Implementation:-

Code:- (main code has been bolded)

#include<stdio.h>

void main()

{

int p,r,i,tp;

printf("\nEnter no. of processes: ");

scanf("%d",&p);

tp=p;

printf("\nEnter no. of resouces: ");

scanf("%d",&r);

int maxi[r];

printf("\nEnter the maximum resources fro each instances: \n");

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

{

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

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

}

int alloc[p][r],max[p][r],avail[r],need[p][r];

int P[p],j;

printf("\nEnter the allocation matrix:");

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

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

printf("\n");

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

{

P[i]=i;

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

for(j=0;j<r;j++)

{

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

}

}

printf("\nEnter the maximum matrix:");

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

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

printf("\n");

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

{

P[i]=i;

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

for(j=0;j<r;j++)

{

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

}

printf("\n");

}

printf("\nEnter the available resouce:\n");

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

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

printf("\n");

for(j=0;j<r;j++)

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

printf("Calculating the NEED matrix...");

printf("\n\n");

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

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

printf("\n");

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

{

P[i]=i;

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

for(j=0;j<r;j++)

{

need[i][j]=max[i][j]-alloc[i][j];

printf("%d",need[i][j]);

}

printf("\n");

}

**int count=0,flag=0,safe[tp],k=0,t,finish[tp];**

**for(i=0;i<tp;i++)**

**{**

**finish[i]=0;**

**}**

**int stack[p][r],maxstack[p][r],Ps[p],dup=0;**

**for(i=0;i<tp;i++)**

**{**

**for(j=0;j<r;j++)**

**{**

**if(need[i][j]>avail[j])**

**break;**

**}**

**if(j!=3)**

**{**

**for(k=0;k<r;k++)**

**{**

**stack[dup][k]=need[i][k];**

**maxstack[dup][k]=max[i][k];**

**}**

**Ps[dup]=i;**

**dup++;**

**}**

**else**

**{**

**safe[count++]=i;**

**printf("\n");**

**printf("P[%d] is executed sucessfully!!!",i);**

**printf("\n");**

**if(j==3){**

**for(k=0;k<r;k++){**

**avail[k]=avail[k]-need[i][k];**

**avail[k]=avail[k]+max[i][k];**

**}**

**}**

**}**

**}**

**int add[dup],diff[dup];**

**for(i=0;i<dup;i++)**

**{**

**add[i]=0;**

**for(j=0;j<r;j++)**

**{**

**diff[i]=maxi[j]-stack[i][j];**

**add[i]=add[i]+diff[i];**

**}**

**}**

**int temp=add[0],swap;**

**for(i=0;i<dup;i++)**

**for(j=i+1;j<dup;j++)**

**{**

**if(add[i]<add[j])**

**{**

**temp=j;**

**}**

**for(k=0;k<r;k++)**

**{**

**swap=stack[i][k];**

**stack[i][k]=stack[temp][k];**

**stack[temp][k]=swap;**

**swap=maxstack[i][k];**

**maxstack[i][k]=maxstack[temp][k];**

**maxstack[i][k]=swap;**

**}**

**temp=Ps[i];**

**Ps[i]=Ps[j];**

**Ps[j]=temp;**

**}**

**for(i=0;i<dup;i++)**

**{**

**for(k=0;k<r;k++){**

**avail[k]=avail[k]-stack[i][k];**

**avail[k]=avail[k]+maxstack[i][k];**

**}**

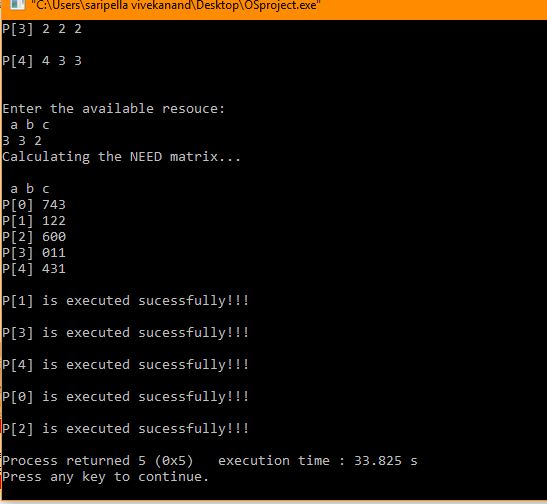
**printf("\nP[%d] is executed successfully!!!",Ps[i]);**

**}**

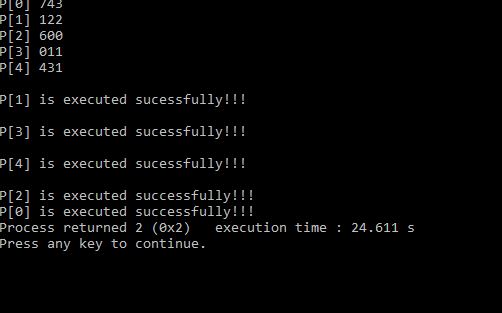
**}**

Results Analysis:-

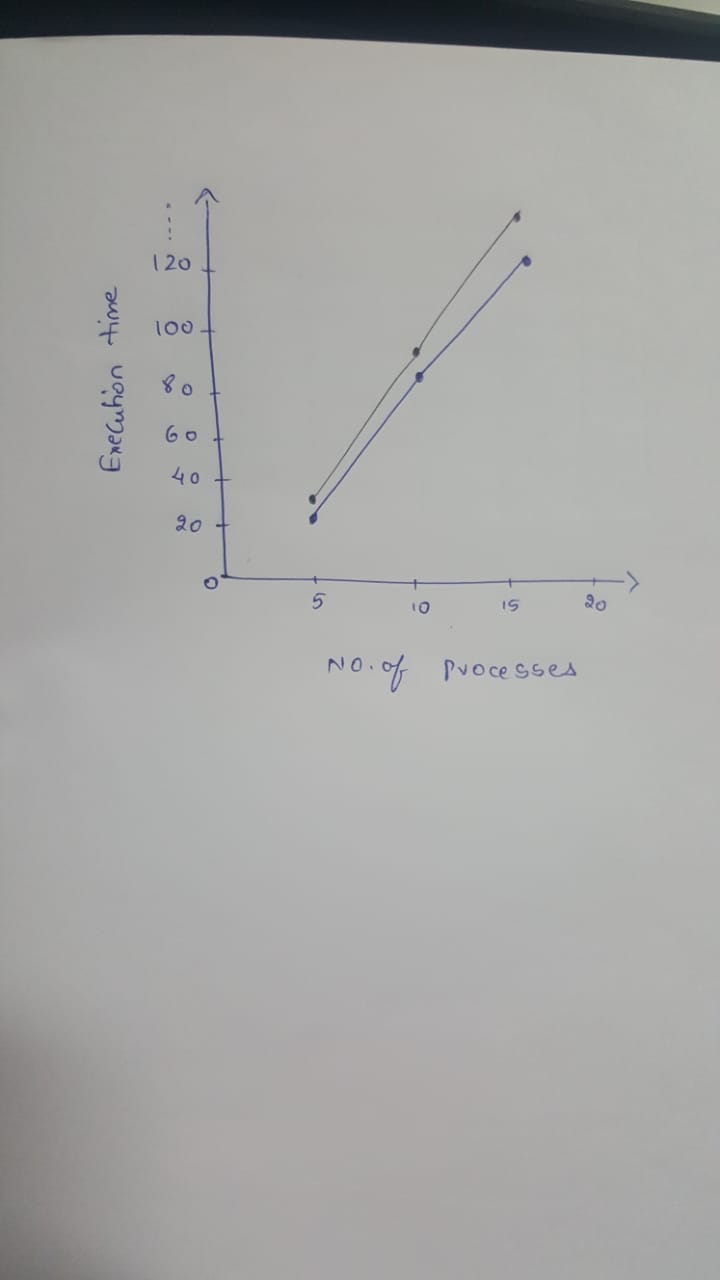
Original bankers algorithm:-



Modified bankers algorithm:-



Graph analysis:-



Conclusion:-

The performance of the proposed concept is better for all ranges because the overhead involved for resource allocation is much lower than that of the Banker’s algorithm . The average improvement of this algorithm over classical algorithm is approximately 9%. This 9% can make a huge difference while dealing with many resources.

Future Work:-

This proposed approach can be used for the killing of undesired process and also for the auto-added process and this area has a bright and great challenges as it processes high demand in coming days.

References:-

1. En.wikipedia.org. (2018). *Banker's algorithm*. [online] Available at: https://en.wikipedia.org/wiki/Banker%27s\_algorithm [Accessed 30 Oct. 2018].
2. Cs.colostate.edu. (2018). *CS 551 Distributed Operating Systems: Banker's Algorithm*. [online] Available at: http://www.cs.colostate.edu/~cs551/CourseNotes/Bankers.html [Accessed 30 Oct. 2018].
3. Personal.kent.edu. (2018). *Operating Systems Notes*. [online] Available at: http://personal.kent.edu/%7Ermuhamma/OpSystems/Myos/deadlockAvoidance.htm [Accessed 30 Oct. 2018].
4. Tsichritzis, D., Bernstein, P. and Rheinboldt, W. (2015). *Operating Systems*. Kent: Elsevier Science.
5. Stallings, W. (1998). *Operating systems*.
6. Haldar, S. and Aravind, A. (2010). *Operating systems*. Delhi: Pearson.
7. McHoes, A. and Flynn, I. (n.d.). *Understanding operating systems*.
8. Fundamentals of operating systems. (2013). [Place of publication not identified]: Springer.
9. Wilkes, M. (1994). Operating systems in a changing world. *ACM SIGOPS Operating Systems Review*, 28(2), pp.9-21.
10. Ng, F., Cheng, C. and Sim, P. (2004). *Operating systems*. Singapore: Perason Prentice Hall.