JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

Lecture-21 By Vivek Dubey

COMPUTER SCIENCE AND ENGINEERING

CS403PC

COMPUTER SCIENCE AND ENGINEERING (ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)

CS403PC

BHARAT INSTITUTE OF ENGINEERING AND TECHNOLOGY

Scheme of Operating System

S. No.	Course Code	Course Title	L	T	P	Credits
3	CS403PC	Operating Systems	3	0	0	3
6	CS406PC	Operating Systems Lab	0	0	3	1.5

Lecture-21 By Vivek Dubey

Aim

Program to implement Deadlock Avoidance & Prevention by using Banker's Algorithm

Concept of Banker's Algorithm

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

Why Banker's algorithm is named so?

- Banker's algorithm is named so because it is used in banking system to check whether loan can be sanctioned to a person or not.
- Suppose there are n number of account holders in a bank and the total sum of their money is S.
- If a person applies for a loan then the bank first subtracts the loan amount from the total money that bank has and if the remaining amount is greater than S then only the loan is sanctioned.
- It is done because if all the account holders comes to withdraw their money then the bank can easily do it.
- In other words, the bank would never allocate its money in such a way that it can no longer satisfy the needs of all its customers. The bank would try to be in safe state always.

Data structures to implement the Banker's Algorithm

- Let **'n'** be the number of processes in the system and **'m'** be the number of resources types.

Available :

- It is a 1-d array of size **'m'** indicating the number of available resources of each type.
- Available[j] = k means there are **'k'** instances of resource type $\mathbf{R_j}$

- **Max :**
  - It is a 2-d array of size '**n*m'** that defines the maximum demand of each process in a system.
  - Max[i, j] = k means process $\mathbf{P_i}$ may request at most **'k'** instances of resource type $\mathbf{R_j}$.

- **Allocation :**
  - It is a 2-d array of size **'n*m'** that defines the number of resources of each type currently allocated to each process.
  - Allocation[i, j] = k means process $\mathbf{P_i}$ is currently allocated **'k'** instances of resource type $\mathbf{R_j}$
- **Need :**
  - It is a 2-d array of size **'n*m'** that indicates the remaining resource need of each process.
  - Need [i, j] = k means process $\mathbf{P_i}$ currently need **'k'** instances of resource type $\mathbf{R_j}$
  - Need [i, j] = Max [i, j] – Allocation [i, j]

Example:

Considering a system with five processes P_0 through P_4 and three resources of type A, B, C.

Resource type A has 10 instances, B has 5 instances and type C has 7 instances.

Suppose at time t_0 following snapshot of the system has been taken.

Considering a system with five processes P_0 through P_4 and three resources of type A, B, C.

Resource type A has 10 instances, B has 5 instances and type C has 7 instances.

Process	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Question1. What will be the content of the Need matrix?

Need [i, j] = Max [i, j] – Allocation [i, j]

So, the content of Need Matrix is:

Process	Need		
	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

Algorithm

- Step 1: Start
- Step 2: Get the values of resources and processes.
- Step 3: Get the avail value.
- Step 4: After allocation find the need value.
- Step 5: Check whether its possible to allocate.
- Step 6: If it is possible then the system is in safe state.
- Step 7: Else system is not in safety state.
- Step 8: If the new request comes then check that the system is in safety or not if we allow request.
- Step 9: Stop

Run

INPUT

```
Enter number of processes –      5
Enter number of resources --     3

Enter details for P0
Enter allocation  --     0    1    0
Enter Max         --     7    5    3
Enter details for P1
Enter allocation  --     2    0    0
Enter Max         --     3    2    2
Enter details for P2
Enter allocation  --     3    0    2
Enter Max         --     9    0    2
Enter details for P3
Enter allocation  --     2    1    1
Enter Max         --     2    2    2
Enter details for P4
Enter allocation  --     0    0    2
Enter Max         --     4    3    3

Enter Available Resources --  3    3    2
Enter New Request Details --
Enter pid --          1
Enter Request for Resources --    1    0    2
```

OUTPUT

```
P1 is visited(      5    3    2)
P3 is visited(      7    4    3)
P4 is visited(      7    4    5)
P0 is visited(      7    5    5)
P2 is visited(      10   5    7)

SYSTEM IS IN SAFE STATE
The Safe Sequence is -- (P1 P3 P4 P0 P2 )
```

Contd..

Process	Allocation		Max	Need		
P0	0	1	0	7	5	3
	7	4	3			
P1	3	0	2	3	2	2
	0	2	0			
P2	3	0	2	9	0	2
	6	0	0			
P3	2	1	1	2	2	2
	0	1	1			
P4	0	0	2	4	3	3
	4	3	1			

Programming Code

- Part-1

```
#include<stdio.h>
struct file
{
int all[10];
int max[10];
int need[10];
int flag;
};
int main()
{
struct file f[10];
int fl;
int i, j, k, p, b, n, r, g, cnt=0, id, newr;
int avail[10],seq[10];
printf("Enter number of processes -- ");
scanf("%d",&n);
printf("Enter number of resources -- ");
scanf("%d",&r);
for(i=0;i<n;i++)
{
printf("Enter details for P%d",i);
printf("\nEnter allocation\t -- \t");
for(j=0;j<r;j++)
scanf("%d",&f[i].all[j]);
printf("Enter Max\t\t -- \t");
for(j=0;j<r;j++)
scanf("%d",&f[i].max[j]);
f[i].flag=0;
}
printf("\nEnter Available Resources\t -- \t");
for(i=0;i<r;i++)
scanf("%d",&avail[i]);
printf("\nEnter New Request Details -- ");
printf("\nEnter pid \t -- \t");
scanf("%d",&id);
printf("Enter Request for Resources \t -- \t");
```

Part-2

```
for(i=0;i<r;i++)
{
scanf("%d",&newr);
f[id].all[i] += newr;
avail[i]=avail[i] - newr;
}
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
f[i].need[j]=f[i].max[j]-f[i].all[j];
if(f[i].need[j]<0)
f[i].need[j]=0;
}
}
```

```
cnt=0;
fl=0;
while(cnt!=n)
{
g=0;
for(j=0;j<n;j++)
{
if(f[j].flag==0)
{
b=0;
for(p=0;p<r;p++)
{
if(avail[p]>=f[j].need[p])
b=b+1;
else
b=b-1; 2
```



```
}
if(b==r)
{
printf("\nP%d is visited",j);
seq[fl++]=j;
f[j].flag=1;
for(k=0;k<r;k++)
avail[k]=avail[k]+f[j].all[k];
cnt=cnt+1;
printf("(");
for(k=0;k<r;k++)
printf("%3d",avail[k]);
printf(")");
g=1;
}
}
}
if(g==0)
{
printf("\n REQUEST NOT GRANTED -- DEADLOCK
OCCURRED"); printf("\n SYSTEM IS IN UNSAFE STATE");
goto y;
}
}
```

Part-3

```
printf("\nSYSTEM IS IN SAFE STATE");
printf("\nThe Safe Sequence is -- (");
for(i=0;i<fl;i++)
printf("P%d ",seq[i]);
printf(")");
y: printf("\nProcess\t\tAllocation\t\tMax\t\t\tNeed\n");

for(i=0;i<n;i++)
{
        printf("P%d\t",i);
        for(j=0;j<r;j++)
        printf("%6d",f[i].all[j]);
        for(j=0;j<r;j++)
        printf("%6d",f[i].max[j]);
        for(j=0;j<r;j++)
        printf("%6d",f[i].need[j]);
        printf("\n");
}
getch();
return 0;
}
```