LOVELY PROFESSIONAL UNIVERSITY

OPERATING SYSTEM PROJECT WORK

SUBMITTED BY : SHIVAM SINGH

REG. NO. : 11708577

EMAIL : 22423.shivam@gmail.com

GITHUB LINK : https://github.com/22423shivam

SUBMITTED TO : RUCHITA DUGGAL

OVERVIEW OF THE CONCEPT

Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.
Consider an example when two trains are coming toward each other on same track and there is only one track, none of the trains can move once they are in front of each other. Similar situation occurs in operating systems when there are two or more processes hold some resources and wait for resources held by other(s). For example, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.1

Deadlock can arise if following four conditions hold simultaneously (Necessary Conditions)
Mutual Exclusion: One or more than one resource are non-sharable (Only one process can use at a time)
Hold and Wait: A process is holding at least one resource and waiting for resources.
No Preemption: A resource cannot be taken from a process unless the process releases the resource.
Circular Wait: A set of processes are waiting for each other in circular form.

Methods for handling deadlock
There are three ways to handle deadlock
1) Deadlock prevention or avoidance: The idea is to not let the system into deadlock state.
2) Deadlock detection and recovery: Let deadlock occur, then do preemption to handle it once occurred.

3) Ignore the problem all together: If deadlock is very rare, then let it happen and reboot the system. This is the approach that both Windows and UNIX take.

Deadlock Detection Algorithm:
The algorithm employs several time varying data structures:

- Available- A vector of length m indicates the number of available resources of each type.
- Allocation- An n*m matrix defines the number of resources of each type currently allocated to a process. Column represents resource and resource represent process.
- Request- An n*m matrix indicates the current request of each process. If request[i][j] equals k then process P_i is requesting k more instances of resource type R_j.

BANKER'S ALGORITHM

The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.
Following Data structures are used 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 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 P_i may request at most 'k' instances of resource type $R_{j.}$

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 P_i is currently allocated 'k' instances of resource type 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 P_i currently need 'k' instances of resource type R_j

for its execution.

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

$Allocation_i$ specifies the resources currently allocated to process P_i and $Need_i$ specifies the additional resources that process P_i may still request to complete its task.
Banker's algorithm consists of Safety algorithm and Resource request algorithm.

MY PROJECT

Ques. 22. Consider following and Generate a solution to find whether the system is in safe state or not?

Available				Processes	Allocation				Max			
A	B	C	D		A	B	C	D	A	B	C	D
1	5	2	0	P0	0	0	1	2	0	0	1	2
				P1	1	0	0	0	1	7	5	0
				P2	1	3	5	4	2	3	5	6
				P3	0	6	3	2	0	6	5	2
				P4	0	0	1	4	0	6	5	6

Ans-

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int need[5][4], completed[5], safeSequence[5];
   int i, j, process, count;
   count = 0;
   int avail[4]={1,5,2,0};
   for(i = 0; i< 5; i++)
      completed[i] = 0;
   int Max[5][4]={0,0,1,2,1,7,5,0,2,3,5,6,0,6,5,2,0,6,5,6};
   int alloc[5][4]={0,0,1,2,1,0,0,0,1,3,5,4,0,6,3,2,0,0,1,4};
```

```
for(i = 0; i < 5; i++)
    for(j = 0; j < 4; j++)
        need[i][j] = Max[i][j] - alloc[i][j];
    do
    {
        printf("\n Max matrix:\tAllocation matrix:\n");
        for(i = 0; i < 5; i++)
        {
            for( j = 0; j < 4; j++)
                printf("%d ", Max[i][j]);
            printf("\t\t");
            for( j = 0; j < 4; j++)
                printf("%d ", alloc[i][j]);
            printf("\n");
        }
        process = -1;
        for(i = 0; i < 5; i++)
        {
            if(completed[i] == 0)
            {
                process = i ;
                for(j = 0; j < 4; j++)
                {
                    if(avail[j] < need[i][j])
                    {
                        process = -1;
                        break;
                    }
                }
            }
```

```
            if(process != -1)
                break;
        }
        if(process != -1)
        {
            printf("\nProcess %d runs to completion!", process);
            safeSequence[count] = process;
            count++;
            for(j = 0; j < 4; j++)
            {
                avail[j] += alloc[process][j];
                alloc[process][j] = 0;
                Max[process][j] = 0;
                completed[process] = 1;
            }
        }
    }
    while(count != 6 && process != -1);
    if(count == 5)
    {
        printf("\nThe system is in a safe state!!\n");
        printf("Safe Sequence : ");
        for( i = 0; i < 5; i++)
            printf("%d-> ", safeSequence[i]);
        printf("\n");
    }
    else
        printf("\nThe system is in an unsafe state!!");
}
```