9th Assignment

Ankita Bonde
19 TEA

Aim: Banker's algorithm for deadlock detection and avoidance

Problem statement: WA Java prog to implement Banker's algorithm.

Theory

① - The banker's algo is a resource allocation and deadlock avoidance algo that test for safety by simulating the allocation of for predetermined max possible amount of all resources. Then makes an "s state" check to test for possible activities before deciding whether allocation should be allowed to continue.

- Following Ds are used to implement Banker's algorithm.

- Let n be the no of process in the system and m be the no of resource types.

Available

- It is a 1-d array of size 'm' indicating the no of available resources of each type
- Available [j] = K means there are K instances of resources type R_j

Max

- It is a 2-d array of size n x m that defines the max demand of each process in a system
- Max [i, j] = K means process P_i may req at most K instances of resource type R_j

Allocation
- It is a 2-d array of size 'n x m' that defines the no of resources of each type currently allocated to each process.
- Allocation [i,j] = K means process P_i is currently allocated K instances of type R_j for execution

Need
- It is a 2-d array of size n x m that indicates remaining resource need of each process
- Need [i,j] = K means P_i currently needs K instances of resource type R_j

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

② Safety algorithm

- The algo for finding out wheter or not a system is in a safe state can be described

① Let work and Finish be vectors of len 'm' and 'n' resp
Initialize : Work = Available
Finish [i] = false : for i = 1, 2, 3, 4 ... n

② Find an i such that both
a] Finish [i] = false
b] Need$_i$ <= Work
if no such i exist goto step 4

③ Work = Work + Allocation [i]
Finish [i] = true
goto step 2

④ if Finish [i] = true for all i
then the system is in a safe state

Example

Process	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	4	3	3			

Ques] Content of need matrix

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

Process	Need		
	A	B	C
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

Q2] Is system in safe state? If yes what is the sequence.

Ans (1) m=3 n=5

Work = available

W = | 3 | 3 | 2 |

	0	1	2	3	4
Finish =	F	F	F	F	F

(2) i = 3

$Need_3$ = 0, 1, 1

Finish$_3$ = false and Need$_3$ < work

So P_3 sequence must be kept in safe seq

③ W = W + A

W = | 7 | 5 | 5 |
A B C

F = | T | T | F | T | T |

④ For i = 4

Need$_4$ = 4 3 1

Finish$_4$ = false and Need$_4$ < Work

So P_4 must be kept in safe sequence

⑤ W = W + A

W = | 7 | 4 | 5 |

F = | 0 F | 1 T | 2 F | 3 T | 4 T |

⑥ For i = 0

Need$_0$ = 7, 4, 3

F[0] is false Need < work

So P_0 must be kept in safe sequence

∴ Safe seq P_1, P_3, P_4, P_0, P_2

Conclusion

Thus we have implemented Banker's algorithm.

```
//Name Ankita Bonde
// TE-A 19
//  ASSINGNMENT 9:GROUP_C_1
// Java program to illustrate Banker's Algorithm

import java.util.*;
class banker_algo
{
static int P = 5;
static int R = 3;

// Function to find the need of each process
static void calculateNeed(int need[][], int maxm[][],
int allot[][])
{
// Calculating Need of each P
for (int i = 0 ; i < P ; i++)
for (int j = 0 ; j < R ; j++)

// Need of instance = maxm instance -
//                 allocated instance
need[i][j] = maxm[i][j] - allot[i][j];
}

// Function to find the system is in safe state or not
static boolean isSafe(int processes[], int avail[], int maxm[][],
int allot[][])
{
int [][]need = new int[P][R];

// Function to calculate need matrix
calculateNeed(need, maxm, allot);

// Mark all processes as infinish
boolean []finish = new boolean[P];

// To store safe sequence
int []safeSeq = new int[P];

// Make a copy of available resources
int []work = new int[R];
for (int i = 0; i < R ; i++)
work[i] = avail[i];

// While all processes are not finished
// or system is not in safe state.
int count = 0;
while (count < P)
{
// Find a process which is not finish and
// whose needs can be satisfied with current
// work[] resources.
```

```
boolean found = false;
for (int p = 0; p < P; p++)
{
// First check if a process is finished,
// if no, go for next condition
if (finish[p] == false)
{
// Check if for all resources of
// current P need is less
// than work
int j;
for (j = 0; j < R; j++)
if (need[p][j] > work[j])
break;

// If all needs of p were satisfied.
if (j == R)
{
// Add the allocated resources of
// current P to the available/work
// resources i.e.free the resources
for (int k = 0 ; k < R ; k++)
work[k] += allot[p][k];

// Add this process to safe sequence.
safeSeq[count++] = p;

// Mark this p as finished
finish[p] = true;

found = true;
}
}
}

// If we could not find a next process in safe
// sequence.
if (found == false)
{
System.out.print("System is not in safe state");
return false;
}
}

// If system is in safe state then
// safe sequence will be as below
System.out.print("System is in safe state.\nSafe"
+" sequence is: ");
for (int i = 0; i < P ; i++)
System.out.print(safeSeq[i] + " ");

return true;
}
```

```
// Driver code
public static void main(String[] args)
{
int processes[] = {0, 1, 2, 3, 4};

// Available instances of resources
int avail[] = {3, 3, 2};

// Maximum R that can be allocated
// to processes
int maxm[][] = {{7, 5, 3},
{3, 2, 2},
{9, 0, 2},
{2, 2, 2},
{4, 3, 3}};

// Resources allocated to processes
int allot[][] = {{0, 1, 0},
{2, 0, 0},
{3, 0, 2},
{2, 1, 1},
{0, 0, 2}};

// Check system is in safe state or not
isSafe(processes, avail, maxm, allot);
}
}
```

_____OUTPUT_____