

The background features a dark blue gradient with faint, light blue circular patterns. These patterns include concentric circles, dashed lines, and degree markings (e.g., 140, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250, 260) arranged in a circular fashion, suggesting a technical or scientific theme.

BANKER'S ALGORITHM

BY

THANAPOL SOPANHARI 6113632

DATA STRUCTURES

- 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.
 - $\text{Available}[j] = k$ 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.
 - $\text{Max}[i, j] = k$ means process P_i may request at most ' k ' instances of resource type R_j .

DATA STRUCTURES

- **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.
 - $\text{Allocation}[i, j] = k$ 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.
 - $\text{Need}[i, j] = k$ means process P_i currently need ' k ' instances of resource type R_j for its execution.
 - $\text{Need}[i, j] = \text{Max}[i, j] - \text{Allocation}[i, j]$

SAFETY ALGORITHM

- 1) Let *Work* and *Finish* be vectors of length '*m*' and '*n*' respectively.
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) $Need_i \leq Work$if no *i* exists go to 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

PROGRAM IMPLEMENTATION

First step is to put your resource and process sizes.

```
Enter resource size: 3
```

```
Enter process size: 5
```

THERE ARE 2 MODES

Manual
mode

Random
mode

MANUAL MODE

PRESS M TO SELECT THE MANUAL MODE

Press M for manual inout or any to random input: **M**

Put resource input:

R1 = **10**

R2 =

5

R3 = **7**

Put allocation input:

0 1 0

2 0 0

3 0 2

2 1 1

0 0 2

Put max input:

7 5 3

3 2 2

9 0 2

2 2 2

4 3 3

EXAMPLE

R0 = 10 R1 = 5 R2 = 7

-----ALLOCATION-----

	R1	R2	R3
P0	0	1	0
P1	2	0	0
P2	3	0	2
P3	2	1	1
P4	0	0	2

-----MAX-----

	R1	R2	R3
P0	7	5	3
P1	3	2	2
P2	9	0	2
P3	2	2	2
P4	4	3	3

-----NEED-----

	R1	R2	R3
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

Available: [3, 3, 2]

-----ALLOCATION-----

	R1	R2	R3
P0	0	1	0
P1	0	0	0
P2	3	0	2
P3	2	1	1
P4	0	0	2

-----NEED-----

	R1	R2	R3
P0	7	4	3
P1	0	0	0
P2	6	0	0
P3	0	1	1
P4	4	3	1

Available:[5, 3, 2]

-----ALLOCATION-----

	R1	R2	R3
P0	0	1	0
P1	0	0	0
P2	3	0	2
P3	0	0	0
P4	0	0	2

-----NEED-----

	R1	R2	R3
P0	7	4	3
P1	0	0	0
P2	6	0	0
P3	0	0	0
P4	4	3	1

Available:[7, 4, 3]

-----ALLOCATION-----

	R1	R2	R3
P0	0	1	0
P1	0	0	0
P2	3	0	2
P3	0	0	0
P4	0	0	0

-----NEED-----

	R1	R2	R3
P0	7	4	3
P1	0	0	0
P2	6	0	0
P3	0	0	0
P4	0	0	0

Available:[7, 4, 5]

-----ALLOCATION-----

	R1	R2	R3
P0	0	0	0
P1	0	0	0
P2	3	0	2
P3	0	0	0
P4	0	0	0

-----NEED-----

	R1	R2	R3
P0	0	0	0
P1	0	0	0
P2	6	0	0
P3	0	0	0
P4	0	0	0

-----ALLOCATION-----

	R1	R2	R3
P0	0	0	0
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	0	0	0

-----NEED-----

	R1	R2	R3
P0	0	0	0
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	0	0	0

Available:[10, 5, 7]

Available vector: R0 = 10 R1 = 5 R2 = 7

The safe-state order: <1>, <3>, <4>, <0>, <2>

RANDOM MODE

PRESS ANY KEYS TO SELECT THE MANUAL MODE

Press M for manual inout or any to random input: *Random*

R0 = 11 R1 = 14 R2 = 17

-----ALLOCATION-----

	R1	R2	R3
P0	1	2	3
P1	2	7	4
P2	1	1	1
P3	4	1	1
P4	3	2	1

-----MAX-----

	R1	R2	R3
P0	4	9	15
P1	7	9	16
P2	2	13	11
P3	6	6	17
P4	4	13	11

-----NEED-----

	R1	R2	R3
P0	3	7	12
P1	5	2	12
P2	1	12	10
P3	2	5	16
P4	1	11	10

Available: [0, 1, 7]

The system is unsafe or deadlock occurs!

<-1> -> <-1> -> <-1> -> <-1> -> <-1>

SAFE ALGORITHM

```
fun isSafe(): Boolean {  
    var isVisited = Array(processSize) { false }  
    var count = 0  
    var work = createAvailable()  
  
    while (count < processSize) {  
        var flag = false  
        for (i in 0 until processSize) {  
            if (!isVisited[i]) {  
                var jTemp = 0  
                for (j in 0 until resourceSize) {  
                    jTemp = j  
                    if (need[i][jTemp] > work[jTemp]) break  
                }  
                if (jTemp == resourceSize-1) {  
                    isVisited[i] = true  
                    flag = true  
                    resultSequence[count] = i  
                    count++  
                    for (j in 0 until resourceSize) {  
                        work[j] += allocation[i][j]  
                        allocation[i][j] = 0  
                        need[i][j] = 0  
                    }  
                    showMatrix("Allocation")  
                    showMatrix("Need")  
                    println("\nAvailable:" + work.toList())  
                }  
            }  
        }  
        if (!flag) {  
            break  
        }  
    }  
}
```

```
    if (count == processSize) {  
        print("\nAvailable vector: ")  
        for (i in 0 until resourceNum.size) {  
            print(" R$i = ${resourceNum[i]} ")  
        }  
        print("\nThe safe-state order: ")  
        var result = "<${resultSequence[0]}>"  
        for (i in 1 until resultSequence.size) {  
            result += ", <${resultSequence[i]}>"  
        }  
        print(result)  
        return true  
    } else {  
        println("The system is unsafe or deadlock occurs!")  
        var result = "<${resultSequence[0]}>"  
        for (i in 1 until resultSequence.size) {  
            result += " -> <${resultSequence[i]}>"  
        }  
        println(result)  
        return false  
    }  
}
```

RANDOM INOUT

```
fun generateMax() { // Each value less than or equal Ri

    for (column in 0 until resourceSize) {
        for (row in 0 until processSize) {
            try {
                max[row][column] = random.nextInt(resourceNum[column]) + 1
            } catch (e: Exception) {
                print(e)
            }
        }
    }
}
```

```
fun generateAllocation() { // Sum of each resource in all process
    var maxInColumn = 0

    for (column in 0 until resourceSize) {
        for (row in 0 until processSize) {
            maxInColumn += max[row][column]
        }
        while (true) {
            var arrayEachResource = Array(processSize) {
                random.nextInt(max[it][column]) + 1
            }
            if (arrayEachResource.sum() <= resourceNum[column]) {
                putArrayInEachResource(arrayEachResource, column)
                break
            }
        }
    }
}
```

RANDOM INPUT

```
fun calculateNeed() {  
    for (row in 0 until processSize) {  
        for (column in 0 until resourceSize) {  
            var needValue = max[row][column] - allocation[row][column]  
            if (needValue < 0) {  
                needValue = 0  
            }  
            need[row][column] = needValue  
        }  
    }  
}
```

The background is a gradient of deep blue and purple, speckled with white dots resembling a starry sky. Overlaid on this are several faint, white geometric patterns. In the top right, there is a large circular scale with degree markings from 0 to 210 and concentric circles. In the bottom right, there are concentric circles with dashed lines and arrows indicating a clockwise direction. In the bottom left, there are also concentric circles with dashed lines and arrows. A small, partial circular scale is visible in the top left corner.

THANK YOU