**OPERATING SYSTEM**

**Course code: CSE 316**

**NAME OF THE STUDENT: KURAPATI SUMANTH**

**Email ID: sumanthkurapati119@gmail.com**

**GitHub:**

**Registration number: 11806337**

**Roll No: B66**

**Section: K18GE**



**School of Computer Science and Engineering**

Lovely Professional University

Phagwara, Punjab (India)

**Problem:**

Consider i that i a i system i has i P i resources i of i same i type. i These i resources i are i shared i by i Q

processes i time i to i time. i All i processes i request i and i release i the i resources i one i at i a i time. i Generate i a i

solution i to i demonstrate i that, i the i system i is i in i safe i state i when i following i conditions i are i satisfied.

Conditions:

1. i Maximum i resource i need i of i each i process i is i between i 1 i and i P.
2. i Summation i of i all i maximum i needs i is i less i than i P+Q

**Description:**

The i given i problem i is i based i upon i checking i the i safe i situation i when i P i resources i of i same i type i are i

shared i by i Q i processes i time i to i time. i We i can i apply i Banker’s i Algorithm i in i this i situation. i

Banker’s i Algorithm i is i a i resource i allocation i and i deadlock i avoidance i algorithm. i This i algorithm i

test i for i safety i simulating i the i allocation i for i predetermined i maximum i possible i amounts i of i all i

resources, i then i makes i an i “s-state” i check i to i test i for i possible i activities, i before i deciding i whether i

allocation i should i be i allowed i to i continue.

In i simple i terms, i it i checks i if i allocation i of i any i resource i will i lead i to i deadlock i or i not, i OR i is i it i safe i

to i allocate i a i resource i to i a i process i and i if i not i then i resource i is i not i allocated i to i that i process. i

Determining i a i safe i sequence(even i if i there i is i only i 1) i will i assure i that i system i will i not i go i into i deadlock.

Banker’s i algorithm i is i generally i used i to i find i if i a i safe i sequence i exist i or i not. i But i here i we i will i

determine i the i total i number i of i safe i sequences i and i print i all i safe i sequences. i

**ALGORITHM:**

**Safety i Algorithm**

The i algorithm i for i finding i out i whether i or i not i a i system i is i in i a i safe i state i can i be i described i as i follows:

1) Let i Work i and i Finish i be i vectors i of i length i ‘m’ i and i ‘n’ i respectively.

1. Initialize: i Work i = i Available
2. Finish[i] i = i false; i for i i i =1, i 2, i 3, i 4….n

2) Find i an i i i such i that i both

1. Finish[i] i = i false
2. Needi i <= i Work

3. if i no i such i i i exists i goto i step i (4)

3) Work i = i Work i + i Allocation[i]

1. Finish[i] i = i true
2. goto i step i (2)

4) if i Finish i [i] i = i true i for i all i i

6. then i the i system i is i in i a i safe i state

**Banker’s i Algorithm:**

* Keep i many i resources i that i satisfy i the i requirement i of i at i least i one i client
* Whenever i a i process i gets i all i its i resources, i it i needs i to i return i them i in i a i restricted i period.
* When i a i process i requests i a i resource, i it i needs i to i wait
* The i system i has i a i limited i number i of i resources
* Advance i feature i for i max i resource i allocation

**PURPOSE i OF i USE:**

Bankers i algorithm i is i used i to i find i out i the i safe i sequence i of i execution i of i processes i which i when i

followed i prevent i system i to i be i in i deadlock i condition. i Banker's i algorithm i is i used i majorly i in i the i

banking i system i to i avoid i deadlock. i It i helps i you i to i identify i whether i a i loan i will i be i given i or i not. Notations i used i in i banker's i algorithms i are i 1) i Available i 2) i Max i 3) i Allocation i 4) i Need Resource i request i algorithm i enables i you i to i represent i the i system i behaviour i when i a i specific i

process i makes i a i resource i request.

Banker's i algorithm i keeps i many i resources i that i satisfy i the i requirement i of i at i least i one i client The i biggest i drawback i of i banker's i algorithm i this i that i it i does i not i allow i the i process i to i change i its i

Maximum i need i while i processing.

**CODE i SNIPPET:**

#include i <stdio.h> #include i <stdlib.h> int i main() { i i i i int i Max[10][10], i need[10][10], i alloc[10][10], i avail[10], i completed[10], i safeSequence[10];

i i i i int i p, i r, i i, i j, i process, i count;

i i i i count i = i 0;

i i i i printf("Enter i the i no i of i processes i : i ");

i i i i scanf("%d", i &p);

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

i i i i i i i i completed[i] i = i 0;

i i i i printf("\n\nEnter i the i no i of i resources i : i ");

i i i i scanf("%d", i &r);

i i i i printf("\n\nEnter i the i Max i Matrix i for i each i process i : i ");

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

i i i i {

i i i i i i i i printf("\nFor i process i %d i : i ", i i i + i 1);

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

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

i i i i }

i i i i printf("\n\nEnter i the i allocation i for i each i process i : i ");

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

i i i i {

i i i i i i i i printf("\nFor i process i %d i : i ",i i + i 1);

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

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

i i i i }

i i i i printf("\n\nEnter i the i Available i Resources i : i ");

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

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

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

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

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

i i i i i i i i do i i i i i i i i {

i i i i i i i i i i i i printf("\n i Max i matrix:\tAllocation i matrix:\n");

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

i i i i i i i i i i i i {

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

i i i i i i i i i i i i i i i i i i i i printf("%d i ", i Max[i][j]); i i i i i i i i i i i i i i i i printf("\t\t"); i i i i i i i i i i i i i i i i for( i j i = i 0; i j i < i r; i j++)

i i i i i i i i i i i i i i i i i i i i printf("%d i ", i alloc[i][j]); i i i i i i i i i i i i i i i i printf("\n");

i i i i i i i i i i i i }

i i i i i i i i i i i i process i = i -1;

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

i i i i i i i i i i i i {

i i i i i i i i i i i i i i i i if(completed[i] i == i 0)//if i not i completed

i i i i i i i i i i i i i i i i {

i i i i i i i i i i i i i i i i i i i i process i = i i i ;

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

i i i i i i i i i i i i i i i i i i i i {

i i i i i i i i i i i i i i i i i i i i i i i i if(avail[j] i < i need[i][j])

i i i i i i i i i i i i i i i i i i i i i i i i {

i i i i i i i i i i i i i i i i i i i i i i i i i i i i process i = i -1; i i i i i i i i i i i i i i i i i i i i i i i i i i i i break;

i i i i i i i i i i i i i i i i i i i i i i i i } i i i i i i i i i i i i i i i i i i i i } i i i i i i i i i i i i i i i i }

i i i i i i i i i i i i i i i i if(process i != i -1)

i i i i i i i i i i i i i i i i i i i i break;

i i i i i i i i i i i i }

i i i i i i i i i i i i if(process i != i -1)

i i i i i i i i i i i i {

i i i i i i i i i i i i i i i i printf("\nProcess i %d i runs i to i completion!", i process i + i 1);

i i i i i i i i i i i i i i i i safeSequence[count] i = i process i + i 1;

i i i i i i i i i i i i i i i i count++;

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

i i i i i i i i i i i i i i i i {

i i i i i i i i i i i i i i i i i i i i avail[j] i += i alloc[process][j];

i i i i i i i i i i i i i i i i i i i i alloc[process][j] i = i 0;

i i i i i i i i i i i i i i i i i i i i Max[process][j] i = i 0;

i i i i i i i i i i i i i i i i i i i i completed[process] i = i 1;

i i i i i i i i i i i i i i i i } i i i i i i i i i i i i } i i i i i i i i }

i i i i i i i i while(count i != i p i && i process i != i -1);

i i i i i i i i if(count i == i p)

i i i i i i i i {

i i i i i i i i i i i i printf("\nThe i system i is i in i a i safe i state!!\n");

i i i i i i i i i i i i printf("Safe i Sequence i : i < i ");

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

i i i i i i i i i i i i i i i i printf("%d i ", i safeSequence[i]); i i i i i i i i i i i i printf(">\n");

i i i i i i i i }

i i i i i i i i else

i i i i i i i i i i i i printf("\nThe i system i is i in i an i unsafe i state!!");

}

**Boundary i condition i of i implemented i code:**

* Number i of i processes i should i be i positive.
* System i must i have i P i resources i of i same i type.
* All i processes i request i and i release i one i at i a i time..
* Summation i of i all i maximum i needs i is i less i than i P+Q.

**Test i cases:**

|  |  |  |
| --- | --- | --- |
| PROCESS | ALLOCATION  R1 i i R2 i i R3 | i i i i i i MAX  R1 i i R2 i i R3 |
| P1 | 0 i i i 1 i i i 0 | 7 i i i 5 i i i 3 |
| P2 | 2 i i i 0 i i i 0 | 3 i i i 2 i i i 2 |
| P3 | 3 i i i 0 i i i 2 | 9 i i i 0 i i i 2 |
| P4 | 2 i i i 1 i i i 1 | 2 i i i 2 i i i 2 |

Total i Resources i are i i R1 i = i 10, i i R2 i = i 5, i i R3 i = i 7

**Case i 1: i** Sequence i will i be

P2--> i P4--> i P1--> i P3

**Case i 2: i** Sequence i will i be

P2--> i P4--> i P3--> i P1

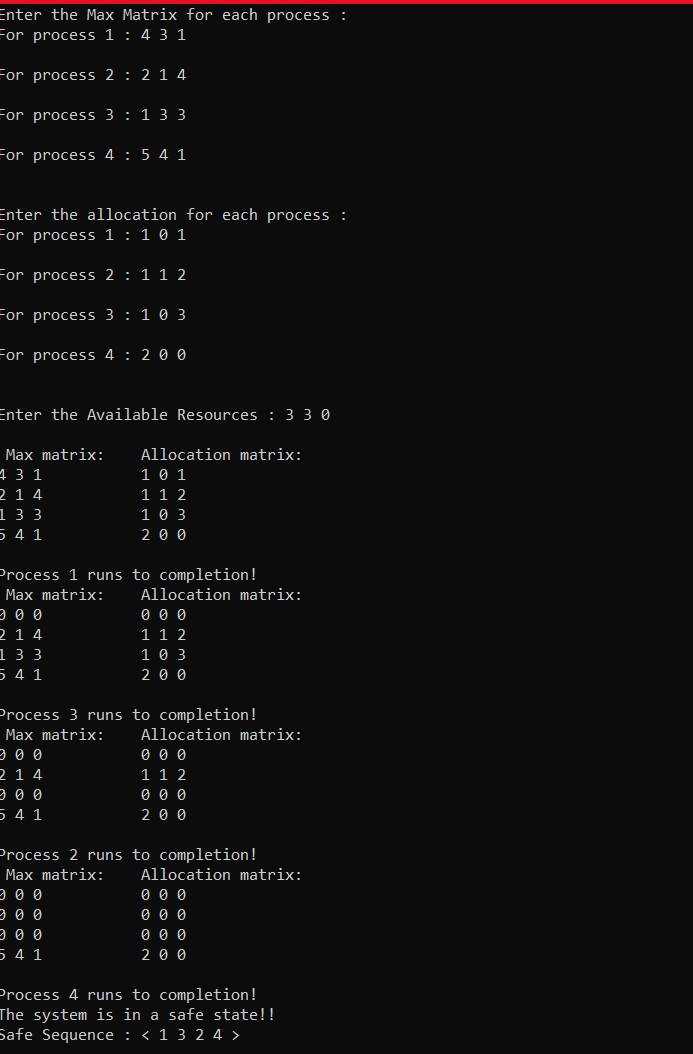
**Case i 3: i** Sequence i will i be

P4--> i P2--> i P1--> i P3

**Case i 4: i** Sequence i will i be

P4--> i P2--> i P3--> i P1

There i are i total i 4 i sequences.



**GITHUB i LINK:**