

EX:7 BANKER'S ALGORITHM

-S.Vishakan CSE-C 18 5001 196

SOURCE CODE:

```
#include <stdio.h>
#include <stdlib.h>

int processes, resources;

void getInput(int instances[resources], int max[processes][resources], int allocated[processes][resources], int need[processes][resources], int available[resources]);

void printTables(int instances[resources], int max[processes][resources], int allocated[processes][resources], int need[processes][resources], int available[resources]);

int processSelector(int need[processes][resources], int available[resources], int completed[processes]);

void safetyAlgorithm(int instances[resources], int max[processes][resources], int allocated[processes][resources], int need[processes][resources], int available[resources]);

int main(void){
    int opt = 0;

    int instances[10];
    int max[10][10];
    int allocated[10][10];
    int need[10][10];
    int available[10];

    while(1){

        printf("\n\n\t\t\tBanker's Algorithm");
        printf("\n\t\t\tMain Menu\n\t1. Read Data\n\t2. Print Data\n\t3. Find A Safe Sequence\n\t0. Exit\n\tYour Option -> ");
        scanf("%d", &opt);
        if(opt == 1){
            printf("\nEnter the number of processes: ");
```

```

        scanf("%d", &processes);
        printf("\nEnter the number of resources: ");
        scanf("%d", &resources);
        getInput(instances, max, allocated, need, available);
    }
    else if(opt == 2){
        printTables(instances, max, allocated, need, available);
    }
    else if(opt == 3){
        safetyAlgorithm(instances, max, allocated, need, available);
    }
    else if(opt == 0){
        printf("\n\t\t\tThank You!");
        break;
    }
    else{
        printf("\n\t\t\tInvalid Option!");
    }
}

return 0;
}

```

```

void getInput(int instances[resources], int max[processes][resources], int allocated[processes]
[resources], int need[processes][resources], int available[resources]){
    int i = 0, j = 0, temp = 0;

```

```

        printf("\nEnter the number of instances of each resource:");
        for(i = 0; i < resources; i++){
            printf("\nResource %d: ", i);
            scanf("%d", &instances[i]);
            available[i] = instances[i];
        }

```

```

        printf("\nEnter the maximum no. of instances of each resource required by each process:
");

```

```

        for(i = 0; i < processes; i++){
            printf("\n\tProcess %d: ", i);
            for(j = 0; j < resources; j++){
                temp = 0;
                while(1){
                    printf("\nResource %d:", j);
                    scanf("%d", &temp);
                    if(temp <= instances[j]){
                        max[i][j] = temp;

```

```

        break;
    }
    else{
        printf("\nMaximum available instances of Resource %d is
%d.", j, instances[j]);
    }
}
}

printf("\nEnter the allocated instances of each resource for each process: ");
for(i = 0; i < processes; i++){
    printf("\n\tProcess %d: ", i);
    for(j = 0; j < resources; j++){
        temp = 0;
        while(1){
            printf("\nResource %d:", j);
            scanf("%d", &temp);
            if(temp <= instances[j]){
                if(temp <= max[i][j]){
                    allocated[i][j] = temp;
                    available[j] -= allocated[i][j];
                    break;
                }
                else{
                    printf("\nMaximum instances of Resource %d
requested by Process %d is %d", j, i, max[i][j]);
                }
            }
            else{
                printf("\nMaximum available instances of Resource %d is
%d.", i, instances[i]);
            }
        }
    }
}

for(i = 0; i < processes; i++){
    for(j = 0; j < resources; j++){
        need[i][j] = max[i][j] - allocated[i][j];
    }
}
}

```

```

void printTables(int instances[resources], int max[processes][resources], int allocated[processes]
[resources], int need[processes][resources], int available[resources]){
    int i = 0, j = 0;

    printf("\nProcess/Resource Table:\n\n");
    printf("\n  %-12s %-12s %-12s %-12s\n  ", "Allocated", "Maximum", "Need",
"Available");

    for(j = 0; j < 4; j++){
        for(i = 0; i < resources; i++){
            printf(" %c ", (65+i));
        }
    }

    for(i = 0; i < processes; i++){
        printf("\nP%d ", i);
        for(j = 0; j < resources; j++){
            printf(" %d ", allocated[i][j]);
        }
        for(j = 0; j < resources; j++){
            printf(" %d ", max[i][j]);
        }
        for(j = 0; j < resources; j++){
            printf(" %d ", need[i][j]);
        }

        if(i == 0){
            for(j = 0; j < resources; j++){
                printf(" %d ", available[j]);
            }
        }
    }
}

```

```

int processSelector(int need[processes][resources], int available[resources], int
completed[processes]){
    int i = 0, j = 0, process = -1, check = 0;

    for(i = 0; i < processes; i++){
        check = 0;
        if(completed[i] == 0){
            for(j = 0; j < resources; j++){
                if(need[i][j] > available[j])
                    check = 1;
            }
        }
    }
}

```

```

        }
    }
    else
        continue;

    if(check == 0) //returning the process if it is not completed and it can be
completed with avl. resources
        return i;
}

if(check == 1){
    return process;    //there is a deadlock
}

if(check == 0){
    return processes+1;    //all processes have completed
}
}

```

```

void safetyAlgorithm(int instances[resources], int max[processes][resources], int
allocated[processes][resources], int need[processes][resources], int available[resources]){
    int deadlock = 0, i = 0, j = 0, process = 0, k = 0, iters = 0;
    int completed[processes];
    int sequence[processes];

    for(i = 0; i < processes; i++){
        completed[i] = 0;
    }

    do{
        process = processSelector(need, available, completed);
        //printf("\nIteration %d: Process Selected : %d", iters, process);
        if(process == -1){
            printf("\nThere is a deadlock!");
            break;
        }

        if(process == processes + 1){
            printf("\nSafe sequence exists!\n");
            for(i = 0; i < processes; i++){
                printf("< P%d ",sequence[i]);
            }
            break;
        }
    }
}

```

```

        completed[process] = 1;           //completing the chosen process
        sequence[k] = process;           //appending it to the safe sequence
        k+=1;

        for(i = 0; i < resources; i++){   //taking back allocated resources
            available[i] += allocated[process][i];
        }

        iters+=1;

    }while(1);
}

```

OUTPUT:

```

PS C:\Users\svish\Desktop> gcc Banker.c -o b
PS C:\Users\svish\Desktop> ./b

```

Banker's Algorithm

Main Menu

- 1. Read Data***
- 2. Print Data***
- 3. Find A Safe Sequence***
- 0. Exit***

Your Option -> 1

Enter the number of processes: 5

Enter the number of resources: 3

Enter the number of instances of each resource:

Resource 0: 10

Resource 1: 10

Resource 2: 10

Enter the maximum no. of instances of each resource required by each process:

Process 0:

Resource 0:7

Resource 1:5

Resource 2:3

Process 1:
Resource 0:3

Resource 1:2

Resource 2:2

Process 2:
Resource 0:9

Resource 1:0

Resource 2:2

Process 3:
Resource 0:2

Resource 1:2

Resource 2:2

Process 4:
Resource 0:4

Resource 1:3

Resource 2:3

Enter the allocated instances of each resource for each process:

Process 0:
Resource 0:0

Resource 1:1

Resource 2:0

Process 1:
Resource 0:2

Resource 1:0

Resource 2:0

Process 2:

Resource 0:3

Resource 1:0

Resource 2:2

Process 3:
Resource 0:2

Resource 1:1

Resource 2:1

Process 4:
Resource 0:0

Resource 1:0

Resource 2:2

Banker's Algorithm
Main Menu

- 1. Read Data*
 - 2. Print Data*
 - 3. Find A Safe Sequence*
 - 0. Exit*
- Your Option -> 2*

Process/Resource Table:

	Allocated	Maximum	Need	Available
	A B C	A B C	A B C	A B C
P0	0 1 0	7 5 3	7 4 3	3 8 5
P1	2 0 0	3 2 2	1 2 2	
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	

Banker's Algorithm
Main Menu

- 1. Read Data*
- 2. Print Data*
- 3. Find A Safe Sequence*
- 0. Exit*

Your Option -> 3

Safe sequence exists!

< P1 < P3 < P0 < P2 < P4

Banker's Algorithm

Main Menu

1. Read Data

2. Print Data

3. Find A Safe Sequence

0. Exit

Your Option -> 0

Thank You!

PS C:\Users\svish\Desktop> gcc Banker.c -o b

PS C:\Users\svish\Desktop> ./b

Banker's Algorithm

Main Menu

1. Read Data

2. Print Data

3. Find A Safe Sequence

0. Exit

Your Option -> 1

Enter the number of processes: 2

Enter the number of resources: 2

Enter the number of instances of each resource:

Resource 0: 3

Resource 1: 3

Enter the maximum no. of instances of each resource required by each process:

Process 0:

Resource 0:3

Resource 1:3

Process 1:

Resource 0:3

Resource 1:3

Enter the allocated instances of each resource for each process:

Process 0:

Resource 0:2

Resource 1:2

Process 1:

Resource 0:1

Resource 1:1

Banker's Algorithm

Main Menu

1. Read Data

2. Print Data

3. Find A Safe Sequence

0. Exit

Your Option -> 2

	Allocated	Maximum	Need	Available
	A B	A B	A B	A B
P0	2 2	3 3	1 1	0 0
P1	1 1	3 3	2 2	

Banker's Algorithm

Main Menu

1. Read Data

2. Print Data

3. Find A Safe Sequence

0. Exit

Your Option -> 3

There is a deadlock!

Banker's Algorithm

Main Menu

1. Read Data

2. Print Data

3. Find A Safe Sequence

0. Exit

Your Option -> 0

Thank You!

