

Name: P.MALREDDY

Reg-No: 192372015

17. Illustrate the deadlock avoidance concept by simulating Banker's algorithm with C.

Aim

The goal of the Banker's Algorithm is to prevent deadlock by allocating system resources to processes in a safe manner, ensuring that no set of processes gets stuck indefinitely waiting for resources.

Banker's Algorithm

1. Input:

- `Available[]`: Number of available instances for each resource.
- `Maximum[][]`: Maximum resource demand for each process.
- `Allocation[][]`: Currently allocated resources for each process.
- `Need[][]`: Remaining resource needs ($Need[i][j] = Maximum[i][j] - Allocation[i][j]$).

2. Steps:

- Compute the `Need` matrix.
- Check if a safe sequence exists:
 - For each process, verify if its resource `Need` can be satisfied using `Available`.
 - If yes, simulate allocation and proceed to the next process.
- If all processes can execute in some sequence without running out of resources, the state is **safe**; otherwise, it's unsafe.

Procedure

1. Input the number of processes and resources.
2. Enter the `Available`, `Maximum`, and `Allocation` matrices.
3. Calculate the `Need` matrix.
4. Run the safety algorithm to find a safe sequence.
5. If a safe sequence is found, display it; otherwise, declare the state unsafe.

Code:

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

```
#include <stdbool.h>
```

```
int main() {  
  
    int n, m, i, j, k;  
  
    printf("Enter number of processes: ");  
  
    scanf("%d", &n);  
  
    printf("Enter number of resources: ");  
  
    scanf("%d", &m);  
  
  
    int Allocation[n][m], Maximum[n][m], Need[n][m], Available[m];  
  
    printf("Enter Allocation matrix:\n");  
  
    for (i = 0; i < n; i++)  
        for (j = 0; j < m; j++)  
            scanf("%d", &Allocation[i][j]);  
  
    printf("Enter Maximum matrix:\n");  
  
    for (i = 0; i < n; i++)  
        for (j = 0; j < m; j++)  
            scanf("%d", &Maximum[i][j]);  
  
    printf("Enter Available resources:\n");  
  
    for (j = 0; j < m; j++)  
        scanf("%d", &Available[j]);  
  
  
    for (i = 0; i < n; i++)
```

```
for (j = 0; j < m; j++)
```

```
    Need[i][j] = Maximum[i][j] - Allocation[i][j];
```

```
bool Finish[n];
```

```
for (i = 0; i < n; i++)
```

```
    Finish[i] = false;
```

```
int SafeSequence[n], work[m];
```

```
for (j = 0; j < m; j++)
```

```
    work[j] = Available[j];
```

```
int count = 0;
```

```
while (count < n) {
```

```
    bool found = false;
```

```
    for (i = 0; i < n; i++) {
```

```
        if (!Finish[i]) {
```

```
            for (j = 0; j < m; j++)
```

```
                if (Need[i][j] > work[j])
```

```
                    break;
```

```
            if (j == m) {
```

```
                for (k = 0; k < m; k++)
```

```
                    work[k] += Allocation[i][k];
```

```

        SafeSequence[count++] = i;

        Finish[i] = true;

        found = true;

    }

}

}

if (!found) {

    printf("System is in an unsafe state.\n");

    return 0;

}

}

printf("System is in a safe state.\nSafe sequence is: ");

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

    printf("%d ", SafeSequence[i]);

printf("\n");

return 0;

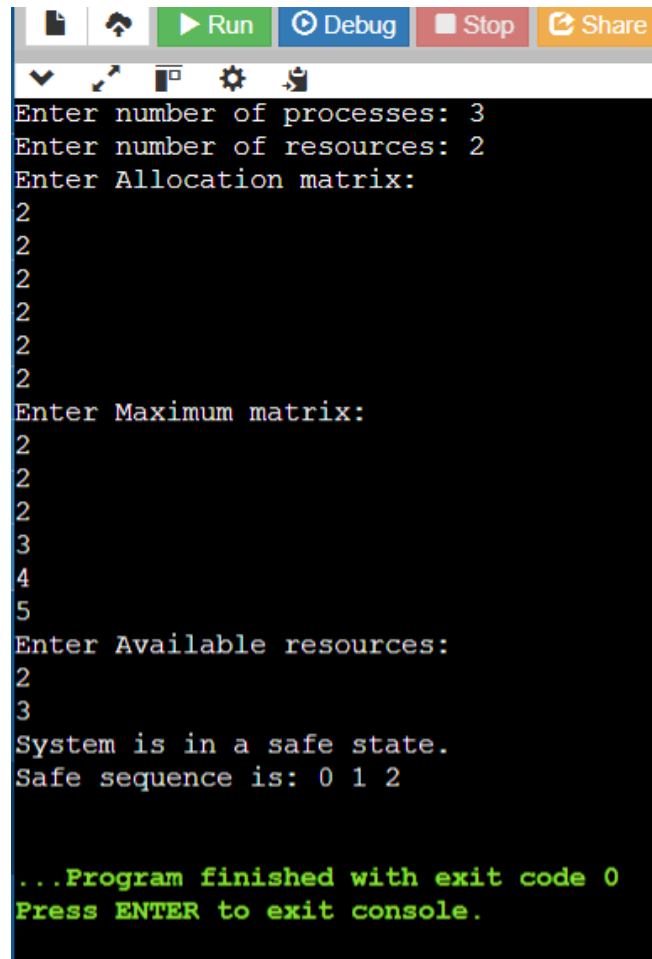
}

```

Result

1. The system checks if a safe sequence exists.
2. If found, it displays the safe sequence (e.g., 0 2 1 3).
3. If not, it reports that the system is in an unsafe state.

Output:



```
Enter number of processes: 3
Enter number of resources: 2
Enter Allocation matrix:
2
2
2
2
2
2
2
Enter Maximum matrix:
2
2
2
3
4
5
Enter Available resources:
2
3
System is in a safe state.
Safe sequence is: 0 1 2

...Program finished with exit code 0
Press ENTER to exit console.
```