**Operating System – CS23431**

| Ex 9 | Dead-Lock Avoidance |
|---|---|
| Name: B M Madhumitha | |
| Reg No: 230701168 | |

**Aim:**

To find out a safe sequence using Banker's algorithm for deadlock avoidance.

**Algorithm:**

1. Initialize work=available and finish[i]=false for all values of i
2. Find an i such that both:
   finish[i]=false and $Need_i$<= work
3. If no such i exists go to step 6
4. Compute work=work+allocationi
5. Assign finish[i] to true and go to step 2
6. If finish[i]==true for all i, then print safe sequence
7. Else print there is no safe sequence

**Program Code:**

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

int main(){
   int n,p;
   printf("\nEnter the no. of resources:");
   scanf("%d",&n);
   printf("\nEnter the no. of processes :");
   scanf("%d",&p);

   int avail[n], need[p][n],max[p][n], alloc[p][n], work[n];
   int finish[p];

   //input for alloc

   // Input available resources
printf("\nEnter the number of available instances for each resource (R1 to R%d):\n", n);
for(int i = 0; i < n; i++) {
   printf("R%d: ", i + 1);
   scanf("%d", &avail[i]);
   work[i] = avail[i];
}

// Initialize finish flags
for(int i = 0; i < p; i++) {
   finish[i] = 0;
}

// Input max matrix
printf("\nEnter the MAX matrix (each row for a process, space-separated values):\n");
```

```
for(int i = 0; i < p; i++) {
    printf("P%d: ", i + 1);
    for(int j = 0; j < n; j++) {
        scanf("%d", &max[i][j]);
    }
}

// Input allocation matrix
printf("\nEnter the ALLOCATION matrix (each row for a process, space-separated values):\n");
for(int i = 0; i < p; i++) {
    printf("P%d: ", i + 1);
    for(int j = 0; j < n; j++) {
        scanf("%d", &alloc[i][j]);
        need[i][j] = max[i][j] - alloc[i][j]; // calculate need matrix inline
    }
}


    int finish_count =1;

    while(true){
        int issafe = 0,comp =0;
        for(int i =0;i<p;i++){
            int flag = 1;
            for(int j=0;j<n;j++){
                if(need[i][j] > work[j]){
                    flag = 0;
                    break;
                }
            }
            if(flag==1 && finish[i]==0){
                for(int j=0;j<n;j++){
                    work[j] = work[j]+alloc[i][j];

                }
                finish[i] = finish_count;
                finish_count++;
                issafe = 1;
            }
            else{
                int allcomp = 1;
                for(int i=0;i<p;i++){
                    if(finish[i]==0)allcomp=0;
                }
                if(allcomp == 1){
                    printf("\nThe safe sequence:");
                    for(int k=1;k<=p;k++){
                        for(int l=0;l<p;l++){
                            if(finish[l]==k)
                            {  printf("P%d",l+1);
                             if (k < p) printf("->");
                               break;
                            }
```

```
                }

            }
            comp =1;
            break;
        }
    }


    }
    if(comp == 1)break;
    if(issafe ==0){
        printf("The Processes Resource allocation is not safe.");
        break;
    }
  }

}
```

Output :

```
C:\Users\kambm\OneDrive\Desktop\Madhumitha\sem IV\OS Assignment\Final version>gcc deadlock_FINAL.c -o deadlock.exe

C:\Users\kambm\OneDrive\Desktop\Madhumitha\sem IV\OS Assignment\Final version>deadlock.exe

Enter the no. of resources:5

Enter the no. of processes :^C
C:\Users\kambm\OneDrive\Desktop\Madhumitha\sem IV\OS Assignment\Final version>deadlock.exe

Enter the no. of resources:3

Enter the no. of processes :5

Enter the number of available instances for each resource (R1 to R3):
R1: 3
R2: 3
R3: 2

Enter the MAX matrix (each row for a process, space-separated values):
P1: 7 5 3
P2: 3 2 2
P3: 9 0 2
P4: 2 2 2
P5: 4 3 3

Enter the ALLOCATION matrix (each row for a process, space-separated values):
P1: 0 1 0
P2: 2 0 0
P3: 3 0 2
P4: 2 1 1
P5: 0 0 2

The safe sequence:P2->P4->P5->P1->P3
C:\Users\kambm\OneDrive\Desktop\Madhumitha\sem IV\OS Assignment\Final version>
```

Result: Thus, the program was executed successfully.