Operating System

Implementation of Banker's Algorithm

By

Shikhar Raj

Using C programming

2 Cases with safe state

and

deadlock

//DOCUMENTATION STARTS HERE

//Name:SHIKHAR RAJ

//T-ID:T00672347

//Details: C program to implement banker's algorithm

//produces a safe state and detect deadlock.

//we need,

//number of process

//number of resources

//initial allocation of instances of resources

//maximum allocation of instances of resources

//step1: first calculate remaining need for each process

//remaining need = max - allocated;

//step2: condition (remaining need <= available instances)

//compare each remaining need with the above condition

//to check which process can be executed first

//then need fits the available, then executed and

//resources released and availability is updated/

//this happens until all resources need is statisfied.

//produces a safe sequence that shows in which order the process

//should be executed in order to prevent a deadlock.

```
//DOCUMENTATION ENDS HERE

//-------------------------------------------------------------------------------------------------
----------------------

//IMPLEMENTATION STARTS HERE

#include<stdio.h>
//variable declarations
int initial_allocation[8][8];
int max_allocation[8][8];
int remaining_need[8][8];
int current_available[8];
int no_of_processes;
int no_of_resources;

//functions
void user_input(int input[8][8])
{
   int x;
   int y;
   for(x=0; x < no_of_processes; x++ )
     for(y=0; y < no_of_resources; y++)
        scanf("%d",&input[x][y]);
}
//-----------------------------------------------------
```

```
void show_input(int input[8][8])
{
   int x;
   int y;
   for(x=0; x < no_of_processes; x++ )
   {
      printf("\n P%d",x);
       for(y=0; y < no_of_resources; y++)
       {
          printf(" %d",input[x][y]);
       }
   }
}
//-------------------------------------------------------
void calc_remaining_need()
{
   int x;
   int y;
   for(x=0; x < no_of_processes; x++ )
     for(y=0; y < no_of_resources; y++)
        remaining_need[x][y] = max_allocation[x][y] - initial_allocation[x][y];
}

//--------CHECK FOR SAFE STATE FUNCTION-----------------------------------

void checkSafeState()
```

```
{
   //initialized variables
   int x = 0;
   int y = 0;
   int z = 0;
   //declare variables
   int flag;
   int end[10];
   int safe_state[10];

   for(x=0; x < no_of_processes; x++ )
   {
      end[x] = 0 ;
   }

   for(x=0; x < no_of_processes; x++ )
   {
      flag = 0;
       if(end[x] == 0)
       {
          for(y=0; y < no_of_resources; y++)
          {
             if(remaining_need[x][y] > current_available[y])
             {
                flag = 1;
                break;
```

```
            }
        }

        if(flag == 0)
        {
            end[x] = 1;
            safe_state[z]=x;
            z++;

            for(y=0; y < no_of_resources; y++)
              current_available[y] += initial_allocation[x][y];
              x = -1;
        }
     }
}

flag = 0;
for(x=0; x < no_of_processes; x++ )
{
   if(end[x] == 0)
   {
     printf("\nDeadlock detected");
     flag = 1;
     break;
   }
}
```

```
   if(flag == 0)
   {
     printf("\nProcesses were executed in a safe sequence:  ");
     for(x=0; x < no_of_processes; x++ )
        printf("P%d=>",safe_state[x]);
   }
}


//--------------main Method-------------------------

int main()
{
   //variables
   int y;

   //user input

      printf("\nInput the number of processes > ");
      scanf("%d",&no_of_processes);
      printf("\nInput the number of resources > ");
      scanf("%d",&no_of_resources);
      printf("\nInput initial_allocation > ");
      user_input(initial_allocation);
      printf("\nInput max_allocation > ");
```

```
    user_input(max_allocation);
    printf("\nInput current_available > ");
    for(y=0; y < no_of_resources; y++)
      scanf("%d",&current_available[y]);


  //show user_input

    printf("\nUSER INPUT");
    printf("\nINTIAL ALLOCATION\n");
    show_input(initial_allocation);

    printf("\n\nMAX ALLOCATION\n");
    show_input(max_allocation);

    printf("\n\nCURRENT AVAILABILITY\n");
    for(y=0; y < no_of_resources; y++)
       printf(" %d",current_available[y]);


  //calculation

    calc_remaining_need();
    printf("\n\nREMAINING NEED\n");
    show_input(remaining_need);
```

```
    //execution
    checkSafeState();

}
```

//IMPLEMENTATION ENDS HERE

//user input screen shots and result screenshots

//Case 1: System is in a safe state – no deadlock

// example from class ppt

```
Input the number of processes > 5

Input the number of resources > 3

Input initial_allocation > 0 1 0 2 0 0 3 0 2 2 1 1 0 0 2

Input max_allocation > 7 5 3 3 2 2 9 0 2 4 2 2 5 3 3

Input current_available >  3 3 2
```

//Output

```
USER INPUT
INTIAL ALLOCATION

 P0 0 1 0
 P1 2 0 0
 P2 3 0 2
 P3 2 1 1
 P4 0 0 2

MAX ALLOCATION

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

CURRENT AVAILABILITY
 3 3 2

REMAINING NEED

 P0 7 4 3
 P1 1 2 2
 P2 6 0 0
 P3 2 1 1
 P4 5 3 1
Processes were executed in a safe sequence:  P1=>P3=>P0=>P2=>P4=>

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

//Case 2: Deadlock detected

// example 2 from class ppt

```
Input the number of processes > 5

Input the number of resources > 4

Input initial_allocation > 0 0 1 2 2 0 0 0 0 0 3 4 2 3 5 4 0 3 3 2

Input max_allocation > 0 0 1 2 2 7 5 0 6 6 5 0 5 3 5 6 0 6 5 2

Input current_available > 2 2 2 2
```

//output

```
USER INPUT
INTIAL ALLOCATION

 P0 0 0 1 2
 P1 2 0 0 0
 P2 0 0 3 4
 P3 2 3 5 4
 P4 0 3 3 2

MAX ALLOCATION

 P0 0 0 1 2
 P1 2 7 5 0
 P2 6 6 5 0
 P3 5 3 5 6
 P4 0 6 5 2

CURRENT AVAILABILITY
 2 2 2 2

REMAINING NEED

 P0 0 0 0 0
 P1 0 7 5 0
 P2 6 6 2 -4
 P3 3 0 0 2
 P4 0 3 2 0
Deadlock detected
```