#### Banker’s Algorithm for Deadlock Avoidance

**Aim:**

To implement the Banker’s algorithm for deadlock avoidance.

**Description:**

Banker's Algorithm is a resource allocation and deadlock avoidance algorithm developed by Edsger Dijkstra. It is named so because it is analogous to the way a bank might manage loan allocation to ensure that it never allocates resources in such a way that it cannot satisfy the maximum possible needs of all its customers (processes). The algorithm helps in determining whether or not a system is in a safe state. A system is in a safe state if there exists a sequence (safe sequence) of all the processes such that each process can obtain its maximum required resources without causing a deadlock.

**Algorithm:**

1. Start
2. Read the number of processes (no\_of\_process)
3. Read the number of resource instances (no\_of\_resources)
4. Initialize arrays for available resources, maximum resources, allocated resources, and needed resources for each process.

**Take Input:**

5. For each process i from 0 to no\_of\_process-1:

* Step 5.1: Read the maximum resources required for each resource type and store them in process[i].max
* Step 5.2: Read the allocated resources for each resource type and store them in process[i].allocated
* Step 5.3: Calculate the needed resources for each resource type as process[i].need[j] = process[i].max[j] - process[i].allocated[j]

1. Read the available resources in the system and store them in the available array.

7. Print the header: "PID\tMaximum\t\tAllocated\tNeed" 8. For each process i from 0 to no\_of\_process-1:

* Step 8.1: Print the process ID i
* Step 8.2: Print the maximum resources required for the process
* Step 8.3: Print the allocated resources for the process
* Step 8.4: Print the needed resources for the process

9. Initialize the work array as a copy of the available resources

10. Initialize the finish array for each process as 0 (indicating not finished)

11. Initialize proceed flag as 1 to start the while loop

12. Initialize k to track the safe sequence

13. While proceed is 1:

* Step 13.1: Set proceed to 0
* Step 13.2: For each process i from 0 to no\_of\_process-1:
  + Step 13.2.1: If finish[i] is 0 (process not finished):
    - Step 13.2.1.1: Check if process[i].need[j] <= work[j] for all j from 0 to no\_of\_resources-1
    - Step 13.2.1.2: If the need can be satisfied:
      * Step 13.2.1.2.1: Update work[j] = work[j] + process[i].allocated[j] for all j
      * Step 13.2.1.2.2: Set finish[i] to 1 (process finished)
      * Step 13.2.1.2.3: Add the process i to the safeSequence
      * Step 13.2.1.2.4: Set proceed to 1 to continue checking

1. Check if all processes are finished:

* Step 14.1: If finish[i] is 1 for all i, the system is in a safe state
* Step 14.2: If any finish[i] is 0, the system is not in a safe state

16. Read the number of processes and resources

17. Declare and initialize the required arrays and structures

18. Call the input function to take inputs

19. Call the showTheInfo function to display the information

20. Check if the system is in a safe state using the isSafeState function:

* Step 20.1: If the system is in a safe state, print the safe sequence
* Step 20.2: If the system is not in a safe state, print that the system is not in a safe state

1. End

**Program:**

#include <stdio.h>

//Ivin Mathew Kurian

//09-05-24

//Bankers algorithm for deadlock avoidance

struct process\_info{

    int max[10];

    int allocated[10];

    int need[10];

};

int no\_of\_process, no\_of\_resources;

void input(struct process\_info process[no\_of\_process], int available[no\_of\_resources]){

    for(int i=0;i<no\_of\_process;i++){

        printf("Enter process[%d] info\n",i);

        printf("Enter Maximum need: ");

        for(int j=0;j<no\_of\_resources;j++){

            scanf("%d",&process[i].max[j]);

        }

        printf("Enter No. of Allocated Resources: ");

        for(int j=0;j<no\_of\_resources;j++){

            scanf("%d",&process[i].allocated[j]);

            process[i].need[j]=process[i].max[j] - process[i].allocated[j];

        }

    }

    printf("Enter Available Resources: ");

    for(int i=0;i< no\_of\_resources;i++){

        scanf("%d",&available[i]);

    }

}

void showTheInfo(struct process\_info process[no\_of\_process]){

    printf("PID\tMaximum\t\tAllocated\tNeed\n");

    for(int i=0;i<no\_of\_process;i++){

        printf("P[%d]\t",i);

        for(int j=0;j<no\_of\_resources;j++){

            printf("%d ",process[i].max[j]);

        }

        printf("\t\t");

        for(int j=0;j<no\_of\_resources;j++){

            printf("%d ",process[i].allocated[j]);

        }

        printf("\t\t");

        for(int j=0;j<no\_of\_resources;j++){

            printf("%d ",process[i].need[j]);

        }

        printf("\n");

    }

}

int isSafeState(struct process\_info process[no\_of\_process],int available[no\_of\_resources],int safeSequence[no\_of\_process]){

    int finish[no\_of\_process];

    int work[no\_of\_resources];

    for(int i=0;i<no\_of\_resources;i++){

        work[i]=available[i];

    }

    for(int i=0;i<no\_of\_process;i++){

        finish[i]=0;

    }

    int proceed=1;

    int k=0;

    while(proceed){

        proceed=0;

        for(int  i=0;i<no\_of\_process;i++){

            int flag=1;

            if(finish[i]==0){

                for(int j=0;j<no\_of\_resources;j++){

                    if(process[i].need[j]<=work[j]){

                        continue;

                    }

                    else{

                        flag=0;

                        break;

                    }

                }

                if(flag==0){

                    continue;

                }

                for(int j=0;j<no\_of\_resources;j++){

                    work[j]=work[j]+process[i].allocated[j];

                }

                finish[i]=1;

                safeSequence[k++]=i;

                proceed=1;

            }

        }

    }

    int i;

    for(i=0;i<no\_of\_process&&finish[i]==1;i++){

        continue;

    }

    if(i==no\_of\_process){

        return 1;

    }

    else{

        return 0;

    }

}

int main()

{

    printf("Enter No of Process\n");

    scanf("%d",&no\_of\_process);

    printf("Enter No of Resource Instances in system\n");

    scanf("%d",&no\_of\_resources);

    int available[no\_of\_resources];

    int safeSequence[no\_of\_process];

    struct process\_info process[no\_of\_process];

    printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Enter details of processes\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

    input(process, available);

    showTheInfo(process);

    if(isSafeState(process, available, safeSequence)){

        printf("System is in safe state\n");

        printf("Safe sequence is: ");

        for(int i=0;i<no\_of\_process;i++){

            printf("P[%d] ",safeSequence[i]);

        }

    }

    else{

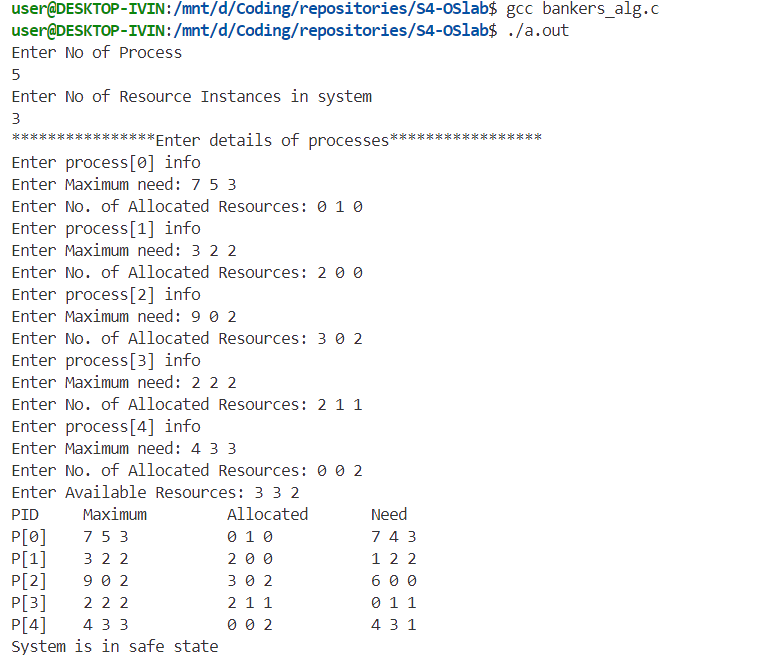
        printf("System is not in Safe State\n");

    }

    return 0;

}

**Output:**



**Result:**

The program has been executed and output has been verified.