#### Deadlock Detection Algorithm

**Aim:**

To implement an algorithm for Deadlock Detection.

**Description:**

The Banker's Algorithm is a resource allocation and deadlock avoidance algorithm used in operating systems. It ensures that processes only request resources within their declared maximum limits, preventing deadlock by checking if a safe sequence of resource allocation exists. The algorithm works by maintaining information about the current allocation and future resource needs of each process and the available resources in the system. It simulates resource allocation to determine if granting a resource request will lead to a safe state, where all processes can eventually complete their execution. If no safe sequence exists for a process, it will be forced to wait until sufficient resources are available, preventing potential deadlock situations. Overall, the Banker's Algorithm contributes to the efficient and safe management of resources in multi-process systems.

**Algorithm:**

**Step 1:** Start.

**Step 2:** Input the number of processes (no\_of\_process) and the number of resource instances (no\_of\_resources).

**Step 3:** Define a structure process\_info to store allocated and requested resources for each process.

**Step 4:** Implement the input function to take user input for allocated and requested resources for each process, and available resources in the system.

**Step 5:** Implement the showTheInfo function to print the allocated and requested resources for each process in tabular form.

**Step 6:** Implement the isSafeState function to determine if the system is in a safe state by checking if a safe sequence exists. Initialize arrays for finish, work, and safeSequence.

**Step 7:** Use a loop to iterate through each process and check if it can be allocated resources based on its request and the available resources.

**Step 8:** If a process can be allocated resources, update the work array and mark the process as finished. Add the process to the safeSequence.

**Step 9:** Continue until all processes are finished or no more processes can be allocated resources.

**Step 10:** Check if all processes are finished. If yes, the system is in a safe state; otherwise, it's not.

**Step 11:** Output whether the system is in a safe state and, if yes, the safe sequence of processes.

**Step 12:** Stop.

**Program:**

#include <stdio.h>

//Ivin Mathew Kurian

//09-05-24

//expt13 Deadlock Detection algorithm

struct process\_info{

    int allocated[10];

    int request[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 No. of Allocated Resources: ");

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

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

        }

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

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

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

        }

    }

    printf("Enter Available Resources: ");

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

        scanf("%d",&available[i]);

    }

}

//Print the Info in Tabular form

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

    printf("PID\tAllocated\tRequest\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].allocated[j]);

        }

        printf("\t\t");

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

            printf("%d ",process[i].request[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].request[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];

    //Create Array of Structure to store Processes's Informations

    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{

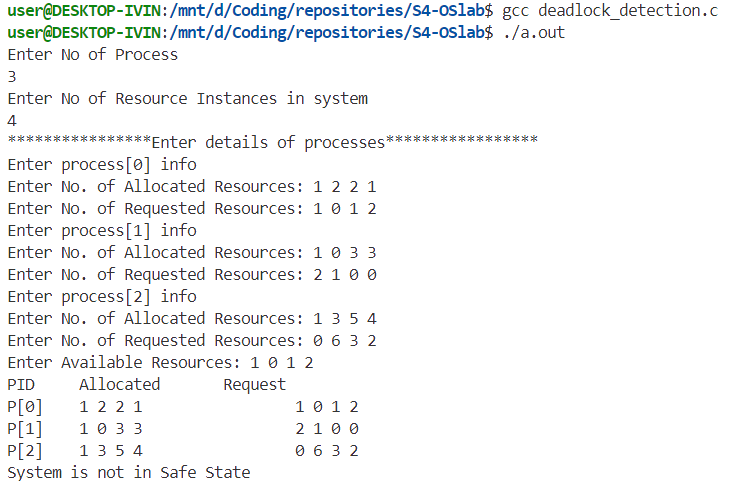
        printf("System is not in Safe State\n");

    }

    return 0;

}

**Output:**



**Result:**

The program has been executed and output has been verified.