Q5a)

Banker's Algorithm with additional resource request checking to grant the request or not.

Aim: To perform Banker's Algorithm with additional resource request checking to grant the request or not.

Algorithm:

BANKER'S ALGORITHM:
1) Let Work and Finish be vectors of length 'm' and 'n' respectively.
   Initialize: Work = Available
   Finish[i] = false; for i=1, 2, 3, 4….n
2) Find an i such that both
   a) Finish[i] = false
   b) $Need_i$ <= Work
   if no such i exists goto step (4)
3) Work = Work + Allocation[i]
   Finish[i] = true
   goto step (2)
4) if Finish [i] = true for all i
   then the system is in a safe state

RESOURCE REQUEST ALGORITHM:
1) If Requesti <= Needi
   Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.
2) If Requesti <= Available
   Goto step (3); otherwise, Pi must wait, since the resources are not available.
3) Have the system pretend to have allocated the requested resources to process Pi by modifying the state as
   follows:
   Available = Available – Requesti
   Allocationi = Allocationi + Requesti
   Needi = Needi– Requesti

Code:

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

void print(int x[][10],int n,int m){
        int i,j;
        for(i=0;i<n;i++){
                printf("\n");
                for(j=0;j<m;j++){
```

```c
                        printf("%d\t",x[i][j]);
                }
        }
}

//Resource Request algorithm
void res_request(int A[10][10],int N[10][10],int AV[10][10],int pid,int m)
{
        int reqmat[1][10];
        int i;
        printf("\n Enter additional request :- \n");
        for(i=0;i<m;i++){
                printf(" Request for resource %d : ",i+1);
                scanf("%d",&reqmat[0][i]);
        }

        for(i=0;i<m;i++)
                if(reqmat[0][i] > N[pid][i]){
                        printf("\n Error encountered.\n");
                        exit(0);
        }

        for(i=0;i<m;i++)
                if(reqmat[0][i] > AV[0][i]){
                        printf("\n Resources unavailable.\n");
                        exit(0);
                }

        for(i=0;i<m;i++){
                AV[0][i]-=reqmat[0][i];
                A[pid][i]+=reqmat[0][i];
                N[pid][i]-=reqmat[0][i];
        }
}

//Safety algorithm
int safety(int A[][10],int N[][10],int AV[1][10],int n,int m,int a[]){

        int i,j,k,x=0;
        int F[10],W[1][10];
        int pflag=0,flag=0;
        for(i=0;i<n;i++)
                F[i]=0;
        for(i=0;i<m;i++)
                W[0][i]=AV[0][i];

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

```c
                for(i=0;i<n;i++){
                        if(F[i] == 0){
                                flag=0;
                                for(j=0;j<m;j++){
                                        if(N[i][j] > W[0][j])
                                                flag=1;
                                }
                                if(flag == 0 && F[i] == 0){
                                        for(j=0;j<m;j++)
                                                W[0][j]+=A[i][j];
                                        F[i]=1;
                                        pflag++;
                                        a[x++]=i;
                                }
                        }
                }
                if(pflag == n)
                        return 1;
        }
        return 0;
}


//Banker's Algorithm
void accept(int A[][10],int N[][10],int M[10][10],int W[1][10],int *n,int *m){
        int i,j;
        printf("\n Enter total no. of processes : ");
        scanf("%d",n);
        printf("\n Enter total no. of resources : ");
        scanf("%d",m);
        for(i=0;i<*n;i++){
                printf("\n Process %d\n",i+1);
                for(j=0;j<*m;j++){
                        printf(" Allocation for resource %d : ",j+1);
                        scanf("%d",&A[i][j]);
                        printf(" Maximum for resource %d : ",j+1);
                        scanf("%d",&M[i][j]);
                }
        }
        printf("\n Available resources : \n");
        for(i=0;i<*m;i++){
                printf(" Resource %d : ",i+1);
                scanf("%d",&W[0][i]);
        }

        for(i=0;i<*n;i++)
                for(j=0;j<*m;j++)
```

```c
                    N[i][j]=M[i][j]-A[i][j];

        printf("\n Allocation Matrix");
        print(A,*n,*m);
        printf("\n Maximum Requirement Matrix");
        print(M,*n,*m);
        printf("\n Need Matrix");
        print(N,*n,*m);

}

int banker(int A[][10],int N[][10],int W[1][10],int n,int m){
        int j,i,a[10];
        j=safety(A,N,W,n,m,a);
        if(j != 0 ){
                printf("\n\n");
                for(i=0;i<n;i++)
                    printf(" P%d  ",a[i]);
                printf("\n A safety sequence has been detected.\n");
                return 1;
        }else{
                printf("\n Deadlock has occured.\n");
                return 0;
        }
}


int main(){
        int ret;
        int A[10][10];
        int M[10][10];
        int N[10][10];
        int W[1][10];
        int n,m,pid,ch;
        printf("\n DEADLOCK AVOIDANCE USING BANKER'S ALGORITHM\n");
        accept(A,N,M,W,&n,&m);
        ret=banker(A,N,W,n,m);
        if(ret !=0 ){
                printf("\n Do you want make an additional request ? (1=Yes|0=No)");
                scanf("%d",&ch);
                if(ch == 1){
                        printf("\n Enter process no. : ");
                        scanf("%d",&pid);
                        res_request(A,N,W,pid-1,m);
                        ret=banker(A,N,W,n,m);
                        if(ret == 0 )
                                exit(0);
```
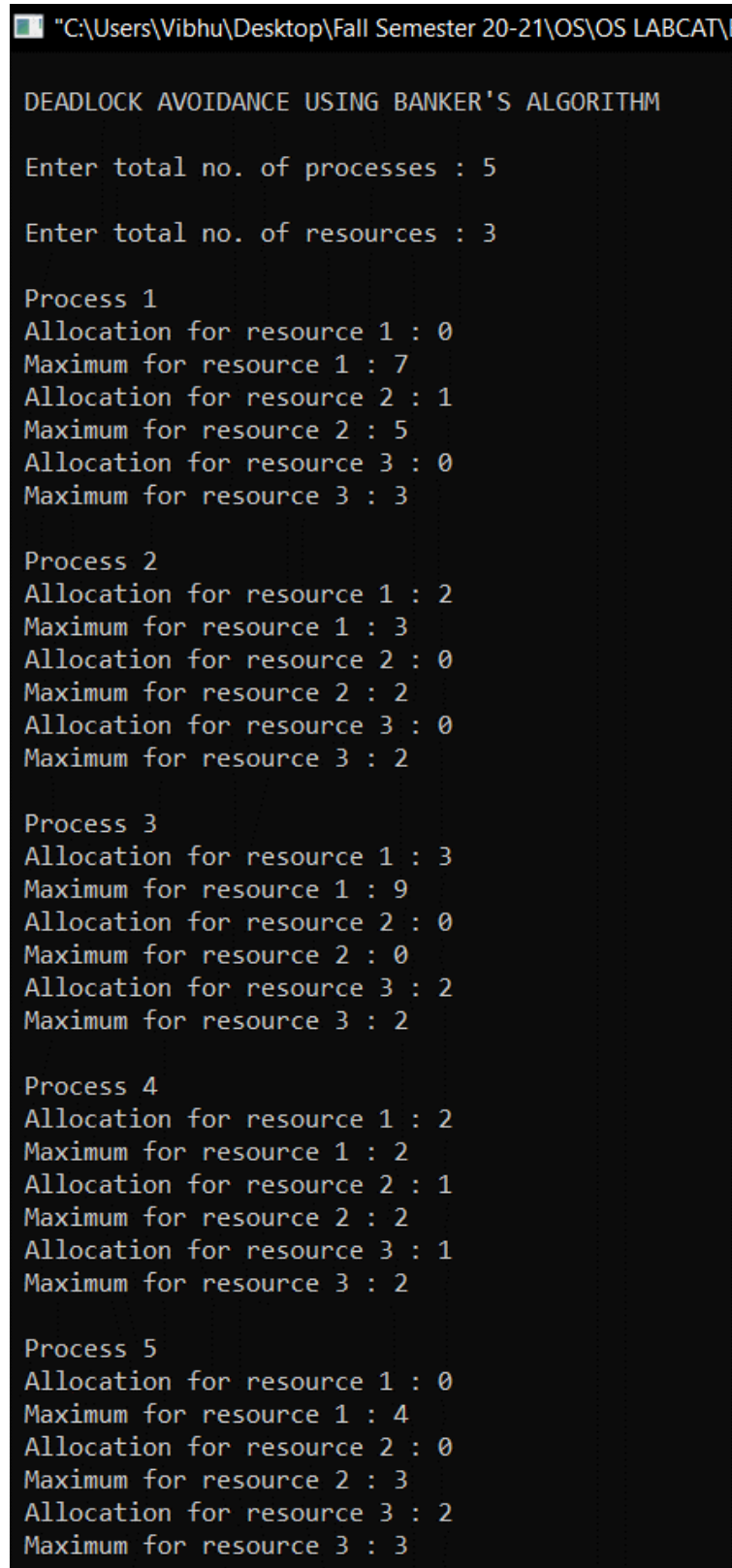
```
            }
    }else
            exit(0);
    return 0;
}
```

Output(screenshots):

```
Available resources :
Resource 1 : 3
Resource 2 : 3
Resource 3 : 2

Allocation Matrix
0        1        0
2        0        0
3        0        2
2        1        1
0        0        2
Maximum Requirement Matrix
7        5        3
3        2        2
9        0        2
2        2        2
4        3        3
Need Matrix
7        4        3
1        2        2
6        0        0
0        1        1
4        3        1

P1   P3   P4   P0   P2
A safety sequence has been detected.

Do you want make an additional request ? (1=Yes|

Enter process no. : 2

Enter additional request :-
Request for resource 1 : 1
Request for resource 2 : 0
Request for resource 3 : 2


P1   P3   P4   P0   P2
A safety sequence has been detected.

Process returned 0 (0x0)   execution time : 115.2
Press any key to continue.
```

Inference: Hence Safety Sequence is detected and Addition Resource Request is granted.

Q5b)

Consider four processes are arrived the ready queue at times 0, 1, 2 and 3 which require 8, 4, 9 and 5 time units to complete their execution. Develop an algorithm and write code to find out the number of context switches are needed, average turnaround time and average waiting time for the shortest job first remaining algorithm.

Aim: To calculate number of context switching, average turn-around time and waiting time in SJF Scheduling algorithm.

Algorithm:

Since it's a little unclear in the question whether we have to take preemptive or non-preemptive mode, I am doing both together:

Non-Preemptive(SJF):
1. Sort all the process according to the arrival time.
2. Then select that process which has minimum arrival time and minimum Burst time.
3. After completion of process make a pool of process which after till the completion of previous process and select that process among the pool which is having minimum Burst time.

Preemptive(SRTF):
1. Traverse until all process gets completely executed.
   a) Find process with minimum remaining time at every single time lap.
   b) Reduce its time by 1.
   c) Check if its remaining time becomes 0
   d) Increment the counter of process completion.
   e) Completion time of current process = current_time +1;
   f) Calculate waiting time for each completed process.
   wt[i]= Completion time - arrival_time-burst_time
   g)Increment time lap by one.
2. Find turnaround time (waiting_time+burst_time).

Code:

SJF(Non-Preemptive):

```c
#include "stdio.h"
#include "stdlib.h"
struct process
{
        int process_id;
        int arrival_time;
        int burst_time;
        int waiting_time;
        int turn_around_time;
};
int main()
{
        int n,i,j;
        int bt=0,k=1,tat=0,sum=0,min;
        printf("Enter number of processes: ");
        scanf("%d",&n);
        struct process proc[n],temp;
        for(i=0;i<n;i++)
        {
                printf("\n");
                printf("Enter arrival time for process%d: ",i+1);
                scanf("%d",&proc[i].arrival_time);
                printf("Enter burst time for process%d: ",i+1);
                scanf("%d",&proc[i].burst_time);
                proc[i].process_id = i+1;
        }

        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                {
                        if(proc[i].arrival_time < proc[j].arrival_time)
                        {
                                temp = proc[j];
                                proc[j] = proc[i];
                                proc[i] = temp;
                        }
                }
        }

        for(i=0;i<n;i++)
        {
                bt+=proc[i].burst_time;
                min = proc[k].burst_time;
                for(j=k;j<n;j++)
```

```
            {
                    if(bt>=proc[j].arrival_time && proc[j].burst_time<min)
                    {
                            temp=proc[k];
                            proc[k]=proc[j];
                            proc[j]=temp;
                    }
            }
            k++;
    }
    proc[0].waiting_time=0;
    int wait_time_total=0;
    int turn_around_time_total=0;
    for(i=1;i<n;i++)
    {
            sum+=proc[i-1].burst_time;
            proc[i].waiting_time = sum-proc[i].arrival_time;
            wait_time_total += proc[i].waiting_time;
    }
    for(i=0;i<n;i++)
    {
            tat+=proc[i].burst_time;
            proc[i].turn_around_time = tat - proc[i].arrival_time;
            turn_around_time_total+=proc[i].turn_around_time;
    }
    printf("Process\tBurst Time\tArrival Time\tWaiting Time\tTurn-Around
Time\n");

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


    printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n",proc[i].process_id,proc[i].burst_time,
proc[i].arrival_time, proc[i].waiting_time,proc[i].turn_around_time);
    }
    printf("Average waiting time: %f\n", (float)wait_time_total/n);
    printf("Average turn around time: %f\n",(float)turn_around_time_total/n);

}
```

<u>SRTF(Preemptive):</u>

```c
#include "stdio.h"
#include "stdlib.h"
struct process
{
        int process_id;
        int arrival_time;
        int burst_time;
        int waiting_time;
        int turn_around_time;
        int remain_time;
};
int main()
{
        int n,i,j;
        int bt=0,k=1,tat=0,sum=0,min;
        printf("Enter number of processes: ");
        scanf("%d",&n);
        struct process proc[n],temp;
        for(i=0;i<n;i++)
        {
                printf("\n");
                printf("Enter arrival time for process%d: ",i+1);
                scanf("%d",&proc[i].arrival_time);
                printf("Enter burst time for process%d: ",i+1);
                scanf("%d",&proc[i].burst_time);
                proc[i].remain_time = proc[i].burst_time;
                proc[i].process_id = i+1;
        }
        int quantum_time,flag=0;
        printf("Enter time quantum: ");
        scanf("%d",&quantum_time);
        int processes_remaining=n;

        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                {
                        if(proc[i].arrival_time < proc[j].arrival_time)
                        {
                                temp = proc[j];
                                proc[j] = proc[i];
                                proc[i] = temp;
                        }
                }
        }
        int wait_time_total=0,totalExecutionTime=0,turn_around_time_total=0;
```

```c
        i=0;
        while(processes_remaining!=0)
        {
                if(proc[i].remain_time<=quantum_time && proc[i].remain_time>0)
                {
                        totalExecutionTime+=proc[i].remain_time;
                        proc[i].remain_time = 0;
                        flag=1;
                }
                else if(proc[i].remain_time>0)
                {
                        proc[i].remain_time-=quantum_time;
                        totalExecutionTime+=quantum_time;
                }
                if(flag==1 && proc[i].remain_time==0)
                {
                        proc[i].waiting_time=totalExecutionTime-proc[i].arrival_time-
proc[i].burst_time;
                        wait_time_total+=proc[i].waiting_time;

                        proc[i].turn_around_time=totalExecutionTime-
proc[i].arrival_time;
                        turn_around_time_total+=proc[i].turn_around_time;
                        flag=0;
                        processes_remaining--;
                }

                if(i==n-1)
                {
                        i=0;
                }
                else if(proc[i+1].arrival_time<=totalExecutionTime)
                        i++;
                else
                        i=0;
        }
        printf("Process\tBurst Time\tArrival Time\tWaiting Time\tTurn-Around
Time\n");

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

        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n",proc[i].process_id,proc[i].burst_time,
proc[i].arrival_time, proc[i].waiting_time,proc[i].turn_around_time);
        }
        printf("Average waiting time: %f\n", (float)wait_time_total/n);
```

```
        printf("Average turn around time: %f\n",(float)turn_around_time_total/n);

    }
```

Output(screenshots):

<u>SJF(Non-Preemptive):</u>

<u>SRTF(Preemptive):</u>

Taking Time Quantum as 3:

The Number of context switches:

                                        p1->p2
                                        p2->p4
                                        p4->p1
                                        p1->p3……..Therefore 4 context switches.