

OPERATING SYSTEMS

LAB CAT

­­

Name: **VIBHU KUMAR SINGH**

Reg. No: **19BCE0215**

Teacher: **Manikandan K.**

**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) Needi <= 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++){

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++){

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++)

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);

}

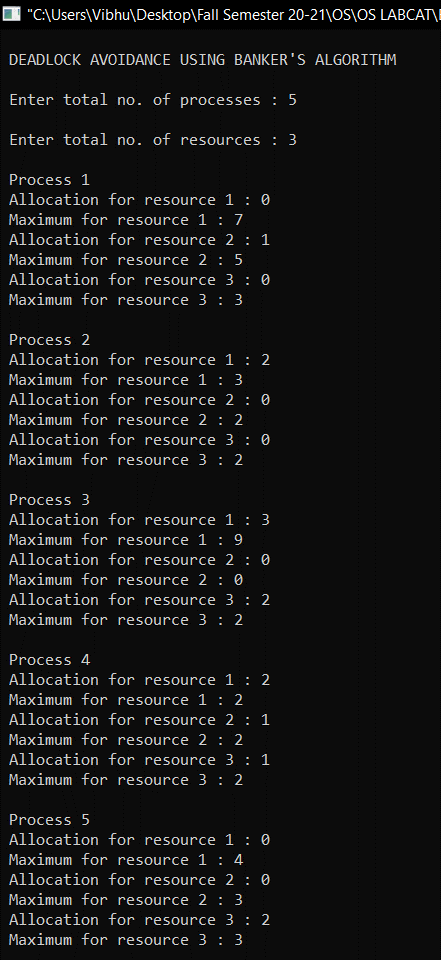
}else

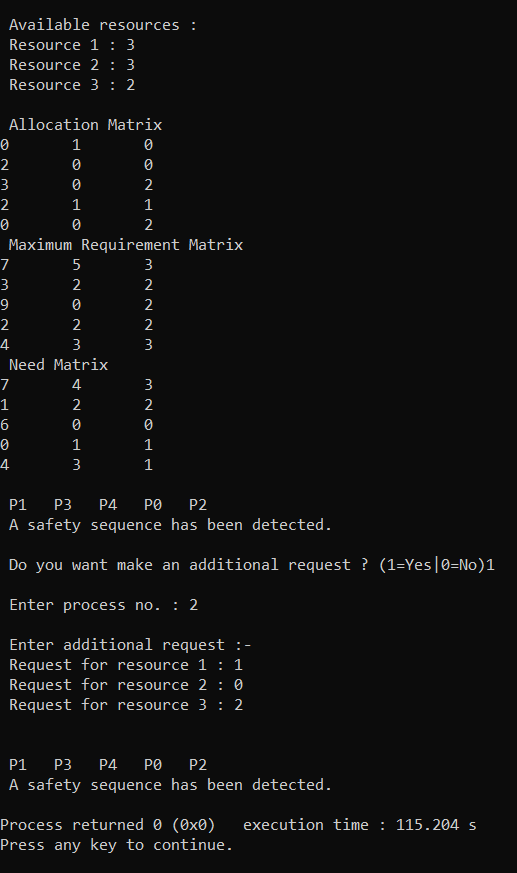
exit(0);

return 0;

}

**Output(screenshots):**





**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):**

#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);

}

**SRTF(Preemptive):**

#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;

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);

}

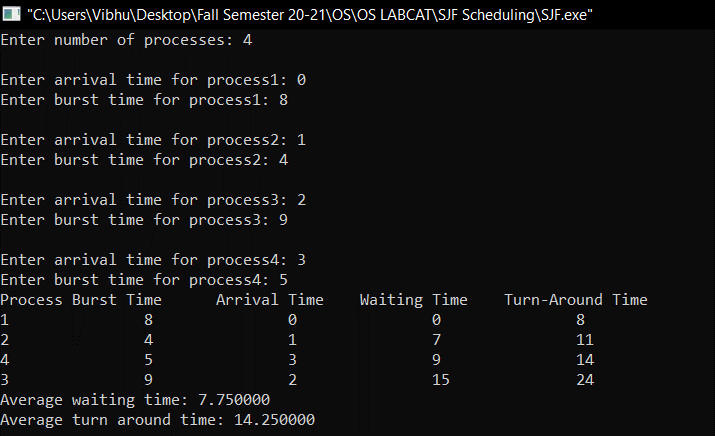
printf("Average waiting time: %f\n", (float)wait\_time\_total/n);

printf("Average turn around time: %f\n",(float)turn\_around\_time\_total/n);

}

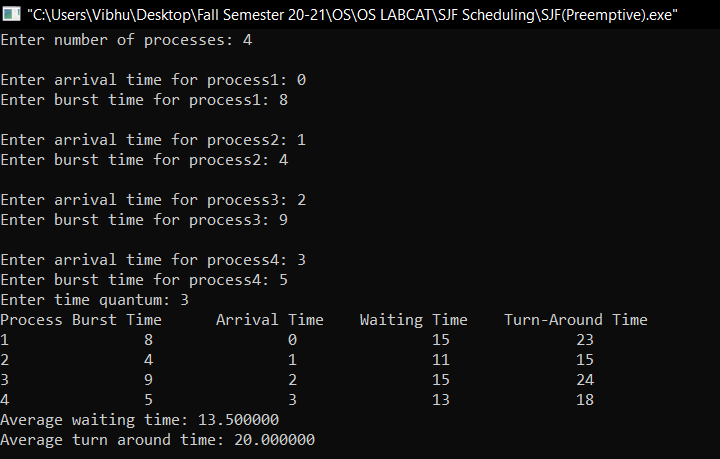
**Output(screenshots):**

**SJF(Non-Preemptive):**



**SRTF(Preemptive):**

Taking Time Quantum as 3:



The Number of context switches:

p1->p2

p2->p4

p4->p1

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