LAB ASSIGNMENT

NAME: TATHAGAT BANERJEE

REG NO.: 17BCE7100

---

7. Write a program in C to implement dining philosopher problem.

Code:

```
#include<stdio.h>
#define n 4
int compltedPhilo = 0,i;
struct fork{
        int taken;
}ForkAvil[n];
struct philosp{
        int left;
        int right;
}Philostatus[n];


void goForDinner(int philID){ //same like threads concept here cases implemented
        if(Philostatus[philID].left==10 && Philostatus[philID].right==10)
       printf("Philosopher %d completed his dinner\n",philID+1);
        //if already completed dinner
        else if(Philostatus[philID].left==1 && Philostatus[philID].right==1)
{
          //if just taken two forks
          printf("Philosopher %d completed his dinner\n",philID+1);
          Philostatus[philID].left = Philostatus[philID].right = 10;
//remembering that he completed dinner by assigning value 10
          int otherFork = philID-1;
```

```
        if(otherFork== -1)
            otherFork=(n-1);
        ForkAvil[philID].taken = ForkAvil[otherFork].taken = 0;
//releasing forks
        printf("Philosopher %d released fork %d and fork
%d\n",philID+1,philID+1,otherFork+1);
        compltedPhilo++;
    }
else if(Philostatus[philID].left==1 && Philostatus[philID].right==0){ //left
already taken, trying for right fork
            if(philID==(n-1)){
                if(ForkAvil[philID].taken==0){ /*KEY POINT OF THIS
PROBLEM, THAT LAST PHILOSOPHER TRYING IN reverse
DIRECTION*/

ForkAvil[philID].taken = Philostatus[philID].right = 1;
printf("Fork %d taken by philosopher %d\n",philID+1,philID+1);
}else{
printf("Philosopher %d is waiting for fork %d\n",philID+1,philID+1);
}
}else{ //except last philosopher case
                int dupphilID = philID;
                philID-=1;
                if(philID== -1)
                    philID=(n-1);

if(ForkAvil[philID].taken == 0){
ForkAvil[philID].taken = Philostatus[dupphilID].right = 1;
printf("Fork %d taken by Philosopher %d\n",philID+1,dupphilID+1);
```

```
}else{
printf("Philosopher %d is waiting for Fork %d\n",dupphilID+1,philID+1);
                }
            }
        }
else if(Philostatus[philID].left==0){ //nothing taken yet
if(philID==(n-1)){
if(ForkAvil[philID-1].taken==0){ /*KEY POINT OF THIS PROBLEM,
THAT LAST PHILOSOPHER TRYING IN reverse DIRECTION*/
ForkAvil[philID-1].taken = Philostatus[philID].left = 1;
printf("Fork %d taken by philosopher %d\n",philID,philID+1);



}else{
printf("Philosopher %d is waiting for fork %d\n",philID+1,philID);
}
}else{ //except last philosopher case
if(ForkAvil[philID].taken == 0){
ForkAvil[philID].taken = Philostatus[philID].left = 1;
printf("Fork %d taken by Philosopher %d\n",philID+1,philID+1);
}else{
printf("Philosopher %d is waiting for Fork %d\n",philID+1,philID+1);
}
}
}else{}
}
int main(){
        for(i=0;i<n;i++)
```

```
        ForkAvil[i].taken=Philostatus[i].left=Philostatus[i].right=0;
        while(compltedPhilo<n){

  /* Observe here carefully, while loop will run until all philosophers
  complete dinner.
  Actually problem of deadlock occur only thy try to take at same time This
  for loop will say that they are trying at same time. And remaining status
  will print by go for dinner function
  */

                for(i=0;i<n;i++)
          goForDinner(i);
printf("\nTill now num of philosophers completed dinner are
%d\n\n",compltedPhilo);
        }
return 0;
}
```

Output:

```
Fork 1 taken by Philosopher 1
Fork 2 taken by Philosopher 2
Fork 3 taken by Philosopher 3
Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 0

Fork 4 taken by Philosopher 1
Philosopher 2 is waiting for Fork 1
Philosopher 3 is waiting for Fork 2
Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 0

Philosopher 1 completed his dinner
Philosopher 1 released fork 1 and fork 4
Fork 1 taken by Philosopher 2
Philosopher 3 is waiting for Fork 2
Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 1

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 2 released fork 2 and fork 1
Fork 2 taken by Philosopher 3
Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 2

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Philosopher 3 released fork 3 and fork 2
Fork 3 taken by philosopher 4

Till now num of philosophers completed dinner are 3

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Fork 4 taken by philosopher 4

Till now num of philosophers completed dinner are 3

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Philosopher 4 completed his dinner
Philosopher 4 released fork 4 and fork 3

Till now num of philosophers completed dinner are 4
```

11. Write a program in C to implement paging techniques.

Code:

```
#include<stdio.h>
void main()
{
int memsize=15;
int pagesize,nofpage;
int p[100];
int frameno,offset;
int logadd,phyadd;
int i;
int choice=0;
printf("\nYour memsize is %d ",memsize);
printf("\nEnter page size:");
scanf("%d",&pagesize);

nofpage=memsize/pagesize;

for(i=0;i<nofpage;i++)
{
printf("\nEnter the frame of page%d:",i+1);
scanf("%d",&p[i]);
}

do
{
printf("\nEnter a logical address:");
scanf("%d",&logadd);
frameno=logadd/pagesize;
offset=logadd%pagesize;
phyadd=(p[frameno]*pagesize)+offset;
printf("\nPhysical address is:%d",phyadd);
printf("\nDo you want to continue(1/0)?:");
scanf("%d",&choice);
}while(choice==1);
}
```

Output:

```
vitap@vitap-OptiPlex-3050:~$ gcc -o 11 11.c
vitap@vitap-OptiPlex-3050:~$ ./11

Your memsize is 15
Enter page size:15

Enter the frame of page1:1

Enter a logical address:12345

Physical address is:0
Do you want to continue(1/0)?:1

Enter a logical address:0

Physical address is:15
Do you want to continue(1/0)?:0
```

12. Implement an algorithm for deadlock detection.

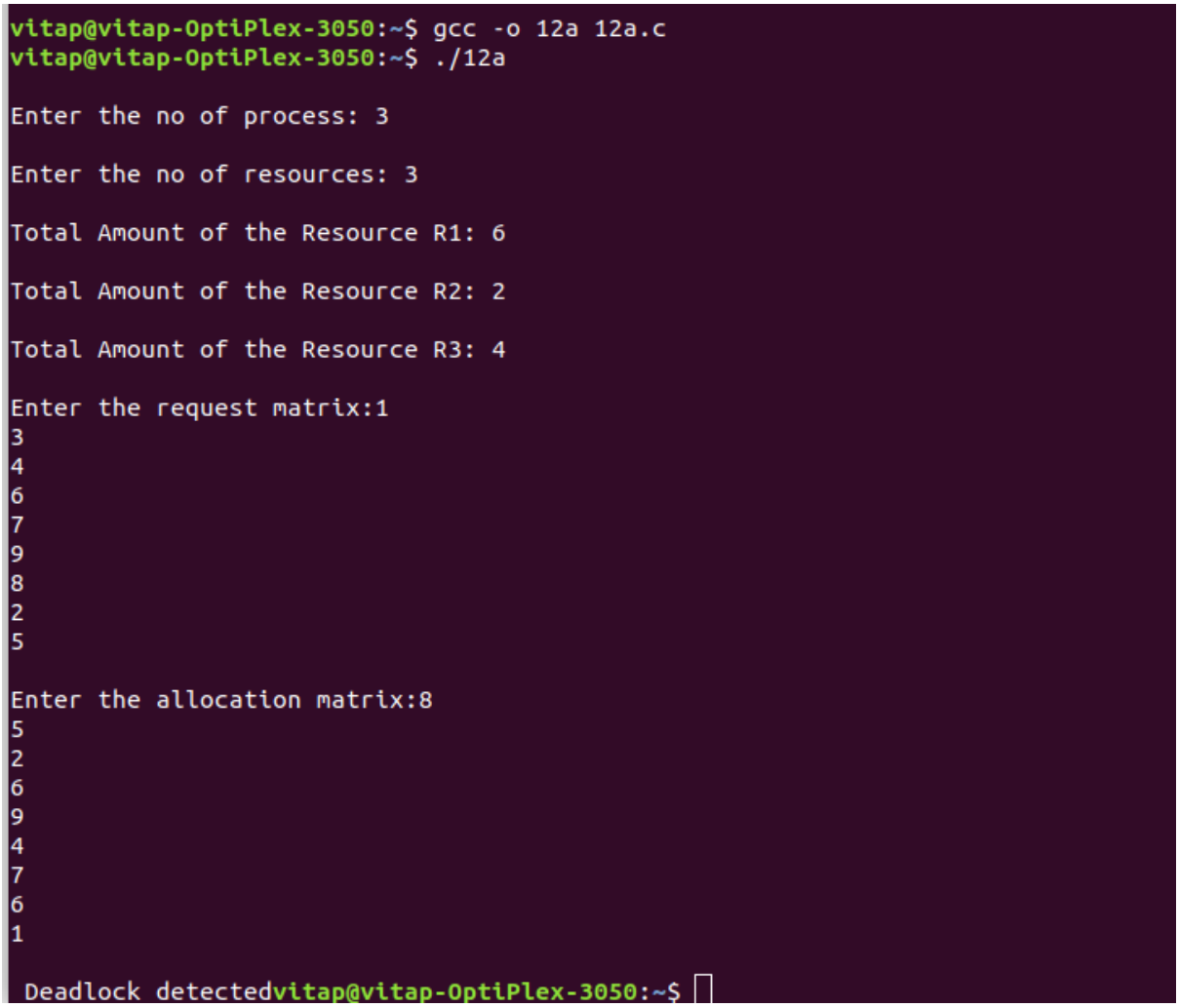
Code:

```
#include<stdio.h>
static int mark[20];
int i,j,np,nr;

int main()
{
int alloc[10][10],request[10][10],avail[10],r[10],w[10];
printf("\nEnter the no of process: ");
scanf("%d",&np);
printf("\nEnter the no of resources: ");
scanf("%d",&nr);
for(i=0;i<nr;i++)
{
printf("\nTotal Amount of the Resource R%d: ",i+1);
scanf("%d",&r[i]);
}

printf("\nEnter the request matrix:");
for(i=0;i<np;i++)
for(j=0;j<nr;j++)
scanf("%d",&request[i][j]);

printf("\nEnter the allocation matrix:");
for(i=0;i<np;i++)
```

```
for(j=0;j<nr;j++)

scanf("%d",&alloc[i][j]);

/*Available Resource calculation*/
for(j=0;j<nr;j++)
{
avail[j]=r[j];
for(i=0;i<np;i++)
{
avail[j]-=alloc[i][j];
}
}

//marking processes with zero allocation
for(i=0;i<np;i++)
{
int count=0;
 for(j=0;j<nr;j++)
   {
     if(alloc[i][j]==0)
       count++;
     else
       break;
   }
 if(count==nr)
 mark[i]=1;
}

// initialize W with avail
for(j=0;j<nr;j++)
    w[j]=avail[j];

//mark processes with request less than or equal to W
for(i=0;i<np;i++)
{
int canbeprocessed=0;
 if(mark[i]!=1)
{
   for(j=0;j<nr;j++)
    {
      if(request[i][j]<=w[j])
```

```
        canbeprocessed=1;
      else
         {
         canbeprocessed=0;
         break;
          }
     }

if(canbeprocessed)
{
mark[i]=1;
for(j=0;j<nr;j++)
w[j]+=alloc[i][j];
}
}
}

//checking for unmarked processes

int deadlock=0;
for(i=0;i<np;i++)
if(mark[i]!=1)
deadlock=1;

if(deadlock)
printf("\n Deadlock detected");

else
printf("\n No Deadlock possible");
}
```

Output:


vitap@vitap-OptiPlex-3050:~$ gcc -o 12a 12a.c
vitap@vitap-OptiPlex-3050:~$ ./12a
Enter the no of process: 3
Enter the no of resources: 3
Total Amount of the Resource R1: 6
Total Amount of the Resource R2: 2
Total Amount of the Resource R3: 4
Enter the request matrix:1
3
4
6
7
9
8
2
5
Enter the allocation matrix:8
5
2
6
9
4
7
6
1
Deadlock detectedvitap@vitap-OptiPlex-3050:~$