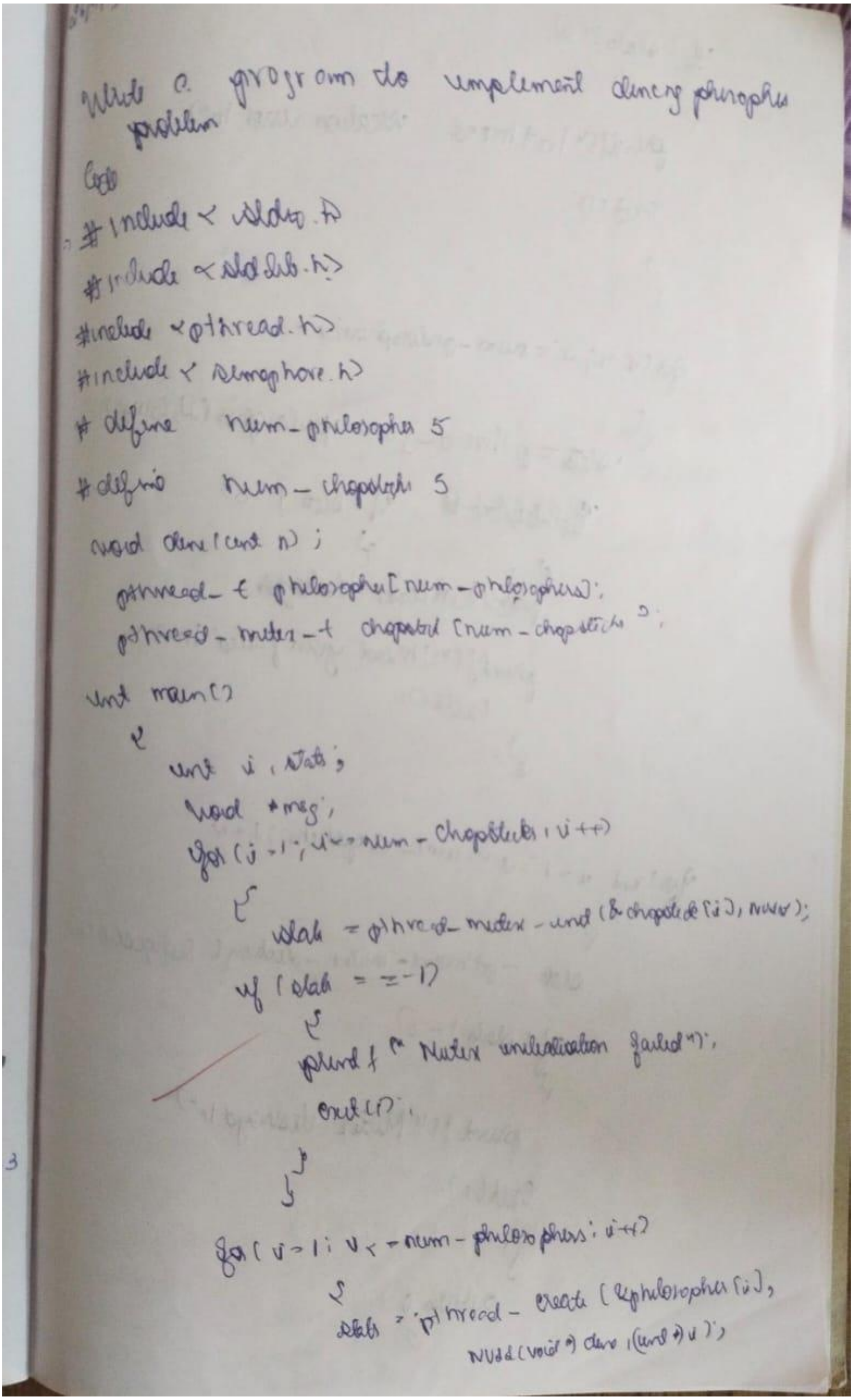
OS LAB-6

Q) **a)Write a C program to simulate the concept of Dining-Philosophers problem.**

b) Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.

CODE-



Write a program to implement dining philosophers problem

Code

```
#include < stdio.h
#include < stdlib.h>
#include <pthread.h>
#include < semaphore.h>
# define    num-philosopher 5
#define     num-chopstick 5
void dine(int n) ;
  pthread-t philosopher[num-philosophers];
  pthread-mutex-t chopstick [num-chopsticks];

int main()
  {
     int i, stats;
      void *msg;
      for (i =1; i<=num-Chopsticks; i++)
        {
            stats = pthread-mutex-init (&chopstick [i], NULL);
         if (stats = =-1)
            {
            printf (" Mutex initialization failed");
             exit(1);
            }
         }
      for ( i=1; i<=num-philosophers; i++)
           {
            stats = pthread- create (&philosopher [i],
                    NULL(void*) dine ,(int*) i );
```

```
if (stats!= 0)
{
  printf("\n thread creation error \n");
  exit(1);
  }
}

for(i=1; i<num-philosophers; i++)
 {
   stats = pthread-join(philosophers[i], &msg);
    ~~[illegible]~~  if(stat != 0)
                   {
     {~~printf("Mutex destroy~~
     printf("Thread join failed \n");
       exit(1);
     }
     }


for(int i=1; i<=num-chopsticks; i++)
     {
     stats = pthread-mutex-destroy( &chopstick[i])
     if (stats != 0)
      {
       printf("Mutex destroyed \n")
      exit(1);
      }
     }
       return 0;
     }
```

```
void dine (int n)
{
  printf(" Philosopher %d is thinking", n)
  pthread_mutex_lock ( & chopstick [n] );
  pthread_mutex_lock ( &chopstick [n+1]%num_
                                        chopstick).
  printf(" Philosopher %d is eating ", n)
  sleep (3);
  pthread_mutex_unlock ( &chopstick[n] );
  pthread_mutex_unlock ( &chopstick (n+1)%.
                              num_chopstick )

  printf ("\n Philosopher %d finished eating ", n)

}}
```

Output

```
Philosopher 1 is thinking
Philosopher 1 is eating
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 3 is eating
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 3 finished eating
Philosopher 1 finished eating
Philosopher 2 is eating
Philosopher 4 is eating
Philosopher 5 is eating
Philosopher 2 finished eating
Philosopher 5 finished eating
Philosopher 4 finished eating
```

10/10

N 2/8/23

· Write a C program to implement the banker's problem using deadlock avoidance system

Code

```
#include<stdio.h>
#include<string.h>
void main()
{
  int alloc[10][10], max[10][10], avail[10], work[10];
int total[10], i, j, k, n, need[10][10];
  int m, count=0, c=0
  char finish[10];
  printf("Enter the no of process & resources\n");
   scanf("%d %d", &n, &m);
   for(i=0; i<=n; i++)
       finish[i]='n';
    printf("enter the clam matrix:\n");
   for(i=0; i<n; i++)
   for(j=0; j<m; j++)
    scanf("%d", &alloc[i][j]);
    printf("Enter the resource vector:");
    for(i=0; i<m; i++)
      scanf("%d", &total[i]);
    for(i=0; i<m; i++)
       avail[i]=0;
        for (j=0; j<m; j++);
        for(i=0; i<n; i++)
        for(j=0; j<m; j++)
               avail[j]+=alloc[i][j];
```

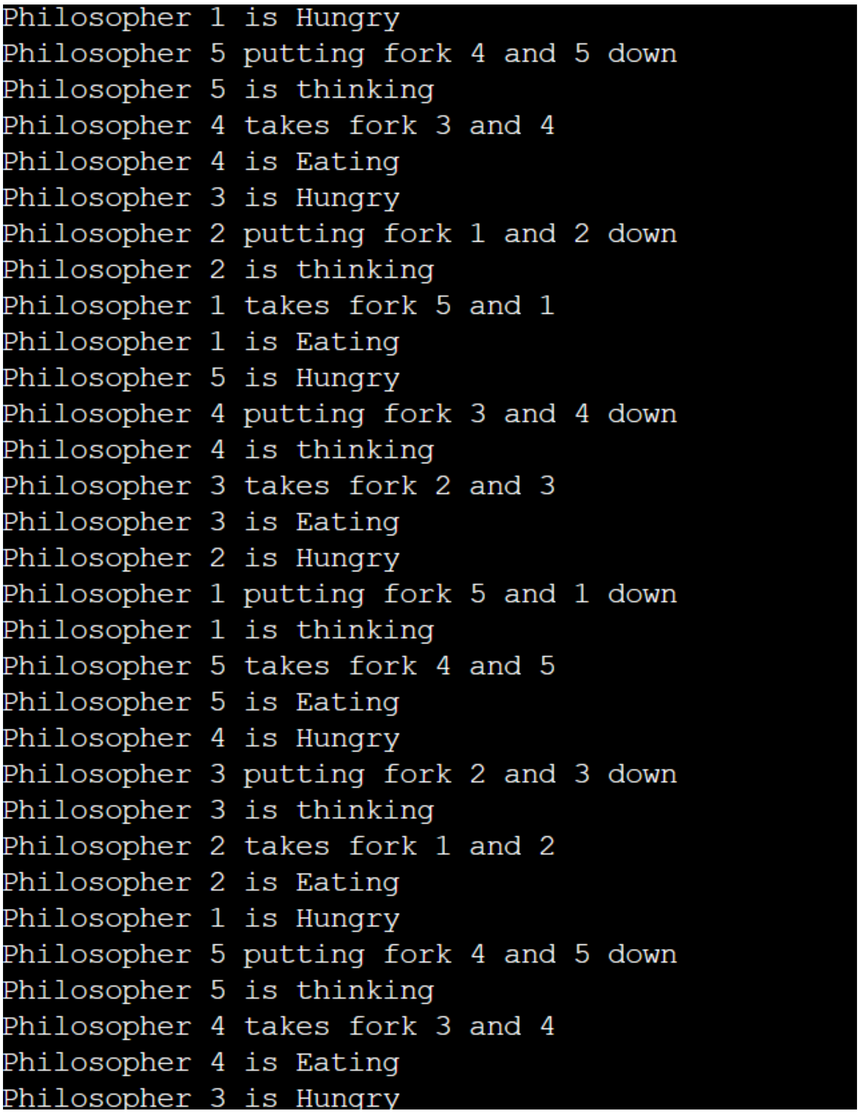
```
for (i = 0; i < m ; i++)
      work[i] = avail[i];

   for(j = 0; j < m; j++)
      work[j] = total[j] - work[j];

   for (i = 0; i < n; i++) {
     for (j = 0 ; j < m; j++) {
     need[i][j] = max[i][j] - alloc[i][j];
     printf(" %d \t", need[i][j]) }
              j++
            printf("\n");
              j = n

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

   {
     c = 0;
    for (j = 0; j < m; j++)
      if ((need[i][j] <= work[j] && finish[i] == 'n'))
        c++;
    if (c == m)

    n    {
    printf(" All the resources can be
              allocated to process %d", i+1);
    printf("\n\n Available resources are: ");
    for (k = 0; k < m; k++)

       {
        work[k] += alloc[i][k];
        printf(" %4d", work[k]);

        }
    printf("\n");
       finish[i] = 'y';
    printf("\n Process %d executed ? : %c
              \n", i + 1, finish[i]); count++;
    }
  }
if (count != n) goto A;
  else
  printf("\n System is in safe mode ");
  printf("\n The given state is safe state ");
  getch();
  }
```

Output

```
   Enter the no of process and
   resources: 4 3
 Enter the claim matrix:
   3  2  2
   6  1  3
   3  1  4
   4  2  2
 Enter the allocation matrix:
   1    0    0
   6    1    2
   2    1    1
 0      0    2
```

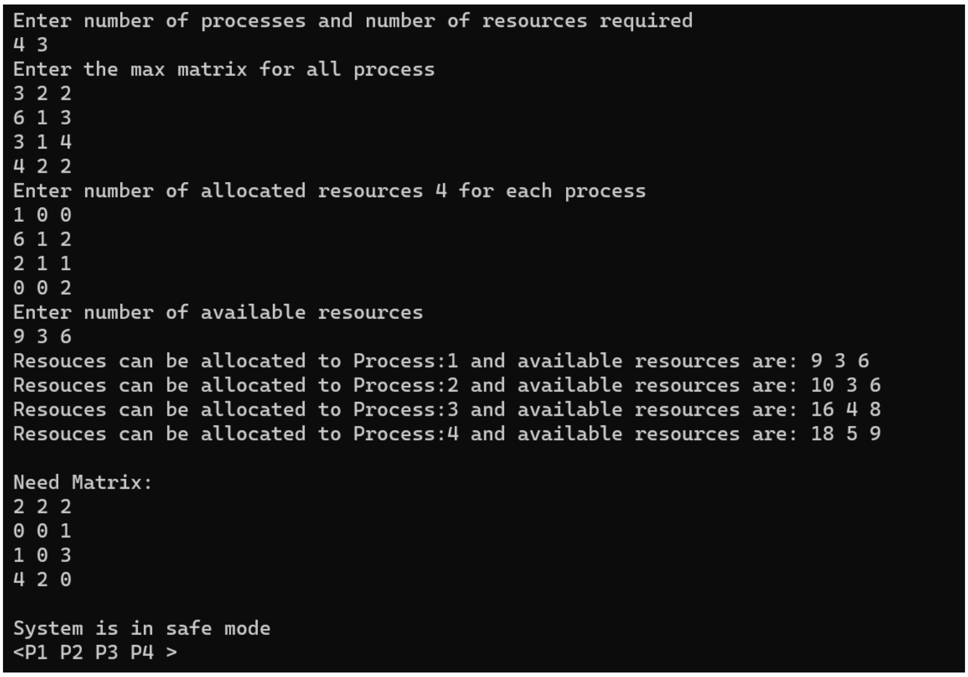
OUTPUT-

Dining-philosopher



```
Philosopher 1 is Hungry
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 3 is Hungry
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 5 is Hungry
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 2 is Hungry
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 4 is Hungry
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 1 is Hungry
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 3 is Hungry
```

Deadlock avoidance



```
Enter number of processes and number of resources required
4 3
Enter the max matrix for all process
3 2 2
6 1 3
3 1 4
4 2 2
Enter number of allocated resources 4 for each process
1 0 0
6 1 2
2 1 1
0 0 2
Enter number of available resources
9 3 6
Resouces can be allocated to Process:1 and available resources are: 9 3 6
Resouces can be allocated to Process:2 and available resources are: 10 3 6
Resouces can be allocated to Process:3 and available resources are: 16 4 8
Resouces can be allocated to Process:4 and available resources are: 18 5 9

Need Matrix:
2 2 2
0 0 1
1 0 3
4 2 0

System is in safe mode
<P1 P2 P3 P4 >
```