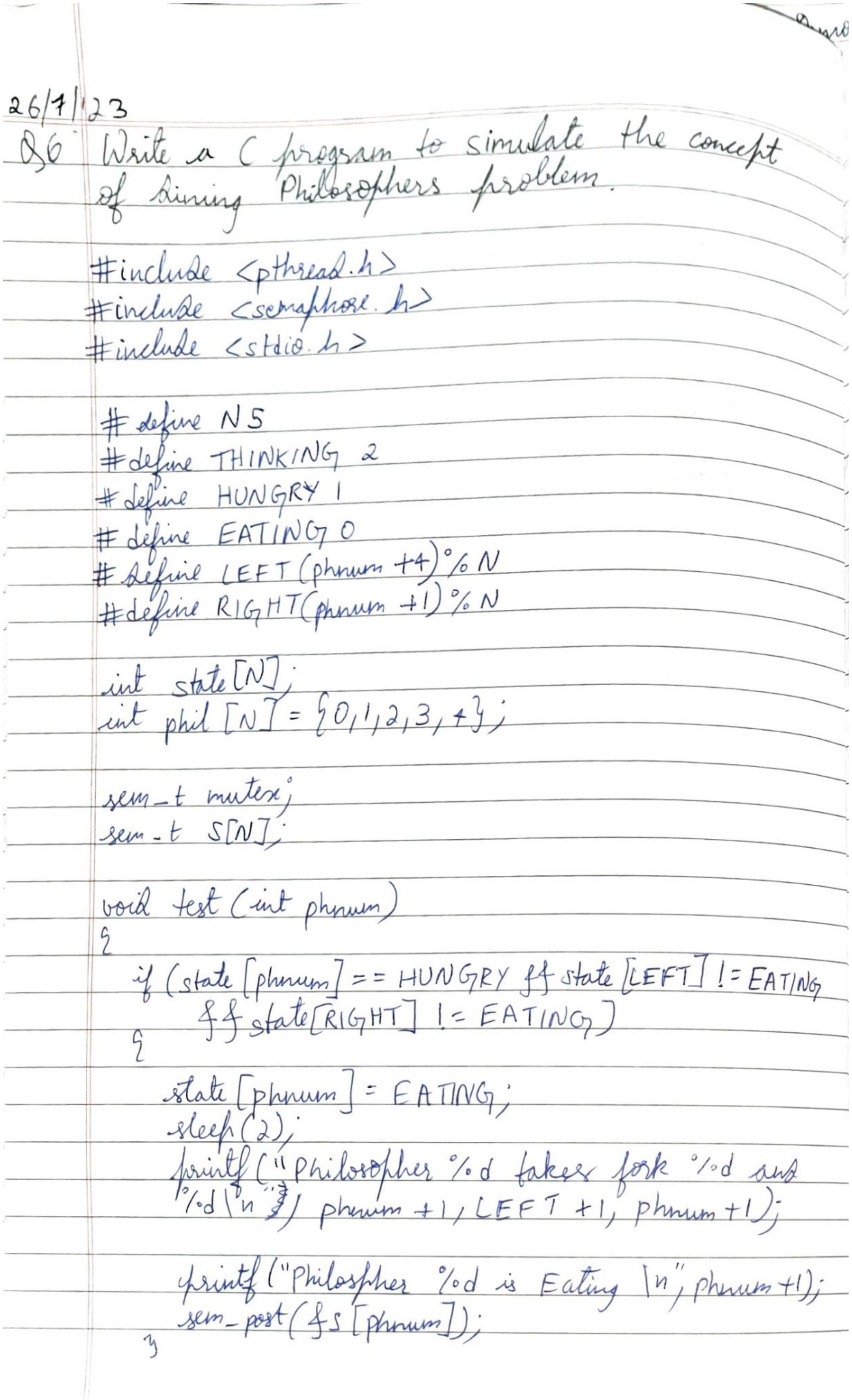
Lab 6: Write a C program to simulate the concept of Dining-Philosophers Problem.



26/7/23

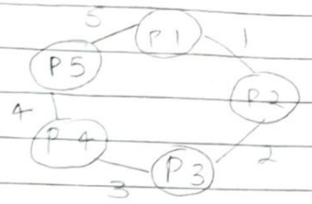
Q6 Write a C program to simulate the concept of dining Philosophers problem.

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>

# define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (phnum +4)%N
#define RIGHT(phnum +1)%N

int state[N];
int phil [N] = {0,1,2,3,4};

sem_t mutex;
sem_t S[N];

void test (int phnum)
{
  if (state[phnum] == HUNGRY && state[LEFT] != EATING
      && state[RIGHT] != EATING)
  {
    state[phnum] = EATING;
    sleep(2);
    printf("Philosopher %d takes fork %d and
    %d \n", phnum +1, LEFT +1, phnum +1);

    printf("Philosopher %d is Eating \n", phnum +1);
    sem_post(&S[phnum]);
  }
}

void take-fork(int phnum)
{
  sem_wait(&mutex);
  state[phnum] =HUNGRY.
  printf("Philosopher %d  is Hungry \n", phnum +1);
  test (phnum);
  sem_post(&mutex);
  sem_wait( &S[phnum]);
  sleep(1);
}

void put_fork(int phnum)
{
  sem_wait (&mutex);

  state[phnum ]=THINKING;

  printf("Philosopher %d putting fork %d and
       %d  down \n"); phnum+1,LEFT+1,
  phnum +1);
  printf("Philosopher %d is thinking\n", phnum +1);

  test (LEFT);
  test (RIGHT);

  sem_post(&mutex);
}
```

```
void* philosopher(void * num)
{
    while (1)
    {
        int *  i = num;
        sleep(1);
        take_fork(*i);
        sleep(0);
        put_fork(*i);
    }
}


int main ()
{
    int i;
    pthread_t thread_id[N];

    sem_init (&mutex ,0,1);

    for(i =0; i < N; i++)
        sem_init[&S[i],0, 0);

    for( i=0; i<N; i++)
    {
     pthread_create(&thread_id[i], NULL, philosopher,
      &phil[i]);
    }

for(i=0; i<N;i++)
    pthread_join (thread_id[i],NULL);
```

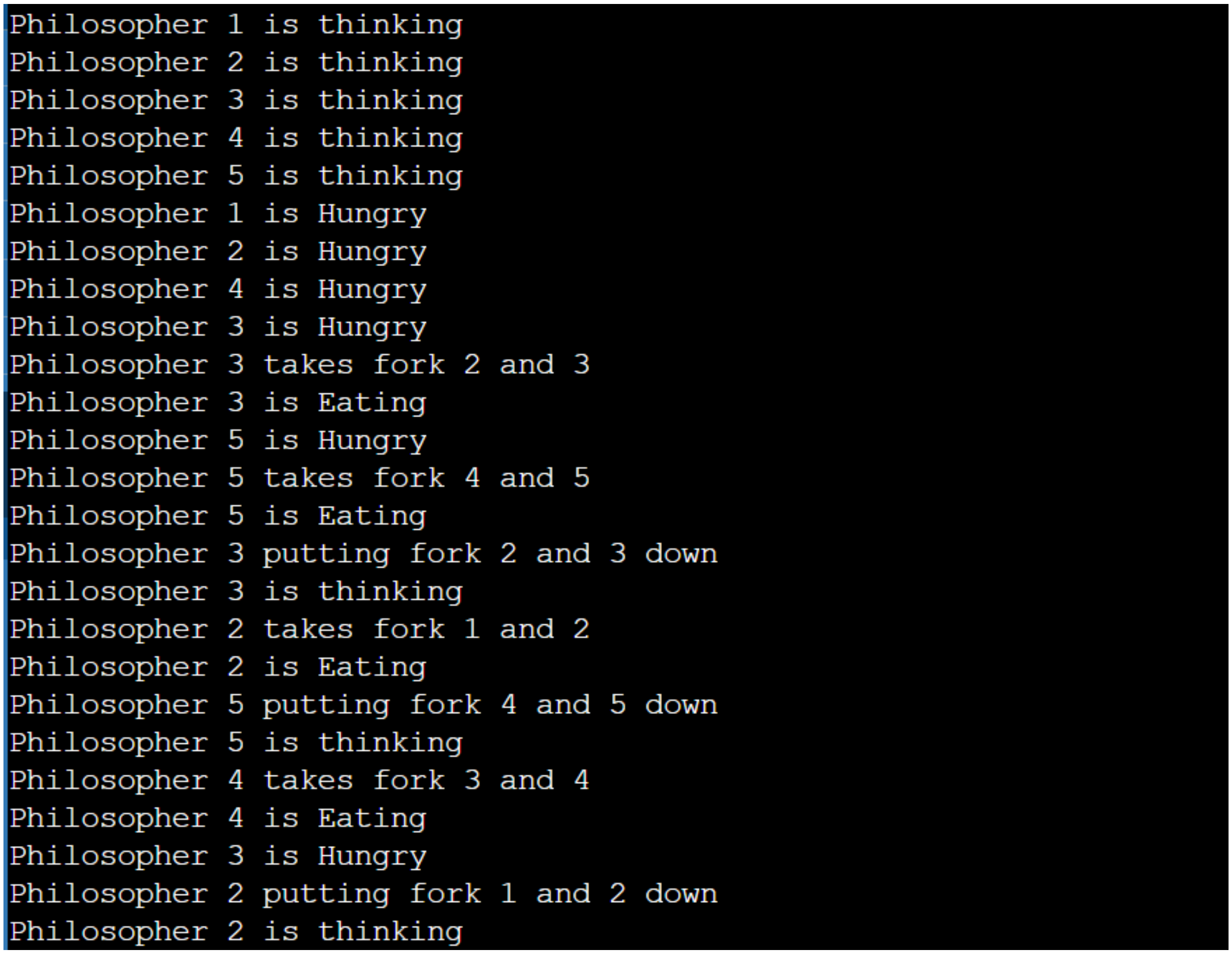
Output :-

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



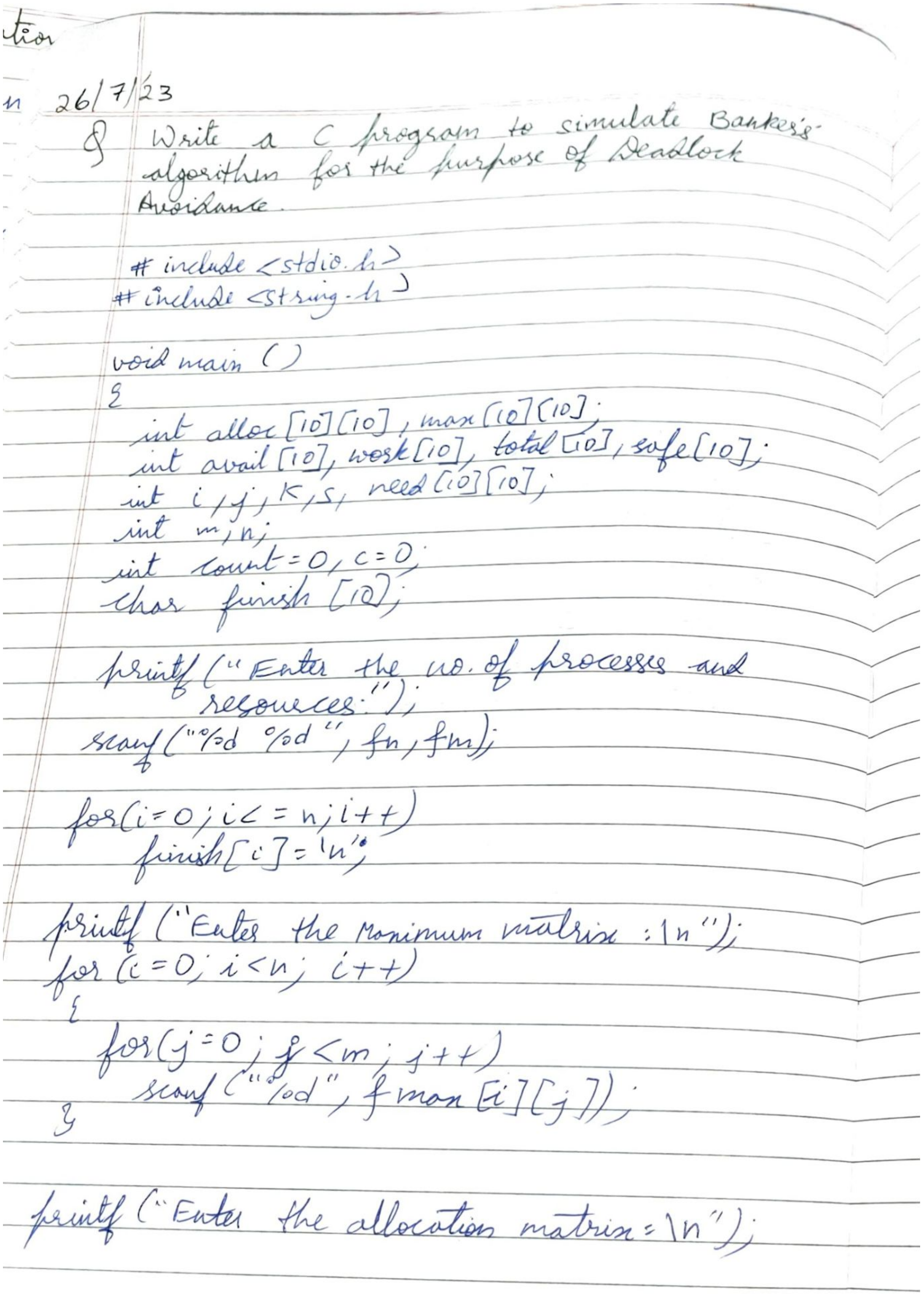
26/7/23

OUTPUT :



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

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



tion

n 26/7/23

8 Write a C program to simulate Banker's algorithm for the purpose of Deadlock Avoidance.

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

void main ()
{
  int alloc[10][10], max[10][10];
  int avail[10], work[10], total[10], safe[10];
  int i,j,k,s, need[10][10];
  int m,n;
  int count=0, c=0;
  char finish[10];

 printf("Enter the no. of processes and
          resources:");
 scanf("%d %d", &n, &m);

for(i=0; i<=n; i++)
     finish[i]='n';

printf("Enter the maximum matrix :\n");
for (i=0; i<n; i++)
{
   for(j=0; j<m; j++)
     scanf("%d", &max[i][j]);
}

printf("Enter the allocation matrix:\n");
```

```
for (i=0; i<n; i++)
{
    for (j=0; j<m; j++)
        scanf ("%d", &alloc[i][j]);
}

printf ("Resource vector: ");

for (i=0; i<m; i++)
    scanf ("%d", &total[i]);

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

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

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

printf ("Need Matrise :\n");

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

```
A :
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)
    {
        printf("All the resources can be allocated
to the 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';
        safe[s++] = i;
        print("\nProcess %d executed? : %c \n\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");
printf("\n The safe sequence is :<");

for(i=0;i<n;i++)
{
    printf("p%d ", safe[i]+1);
}
printf(">");
}
}
```

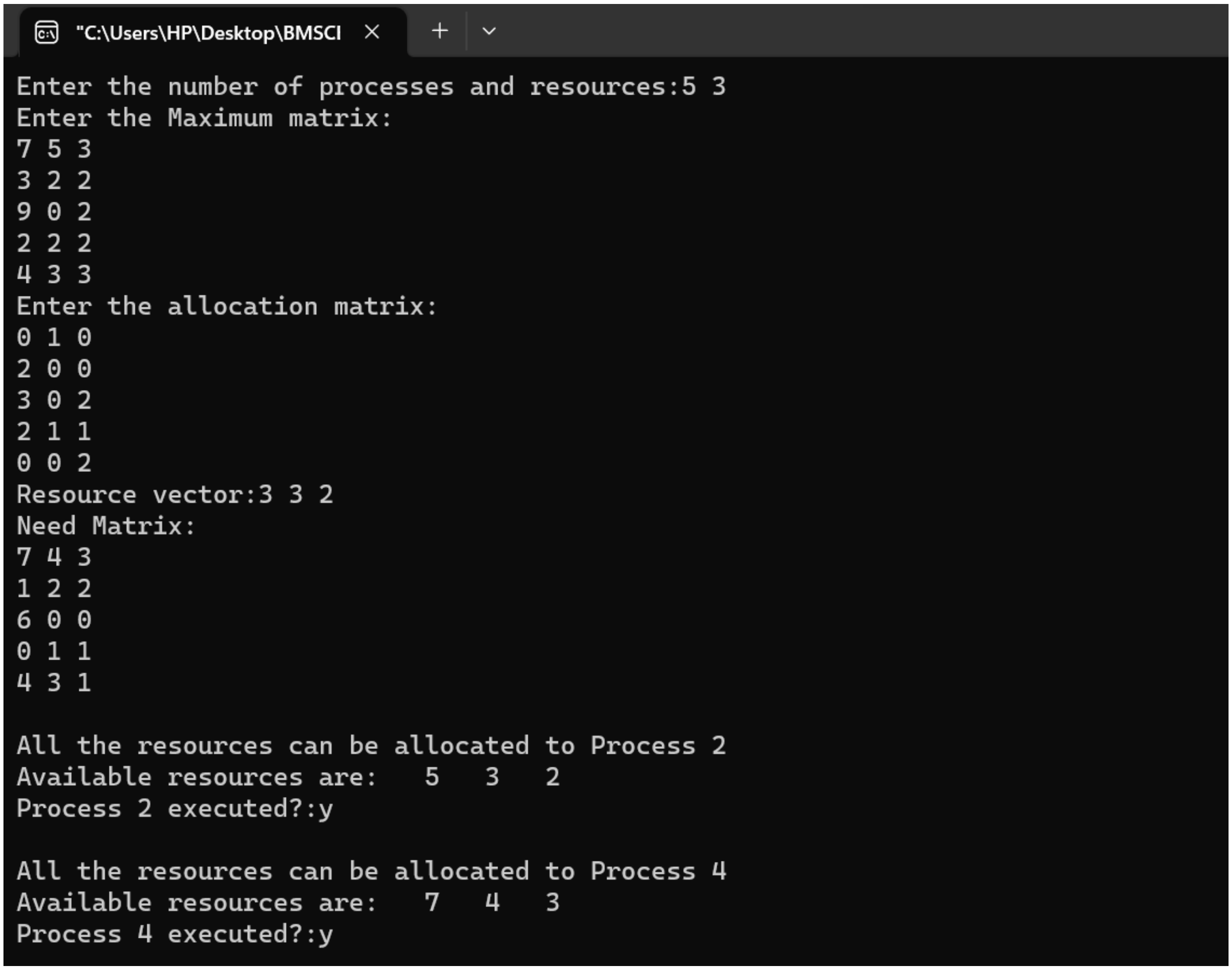
26/7/23

Output:

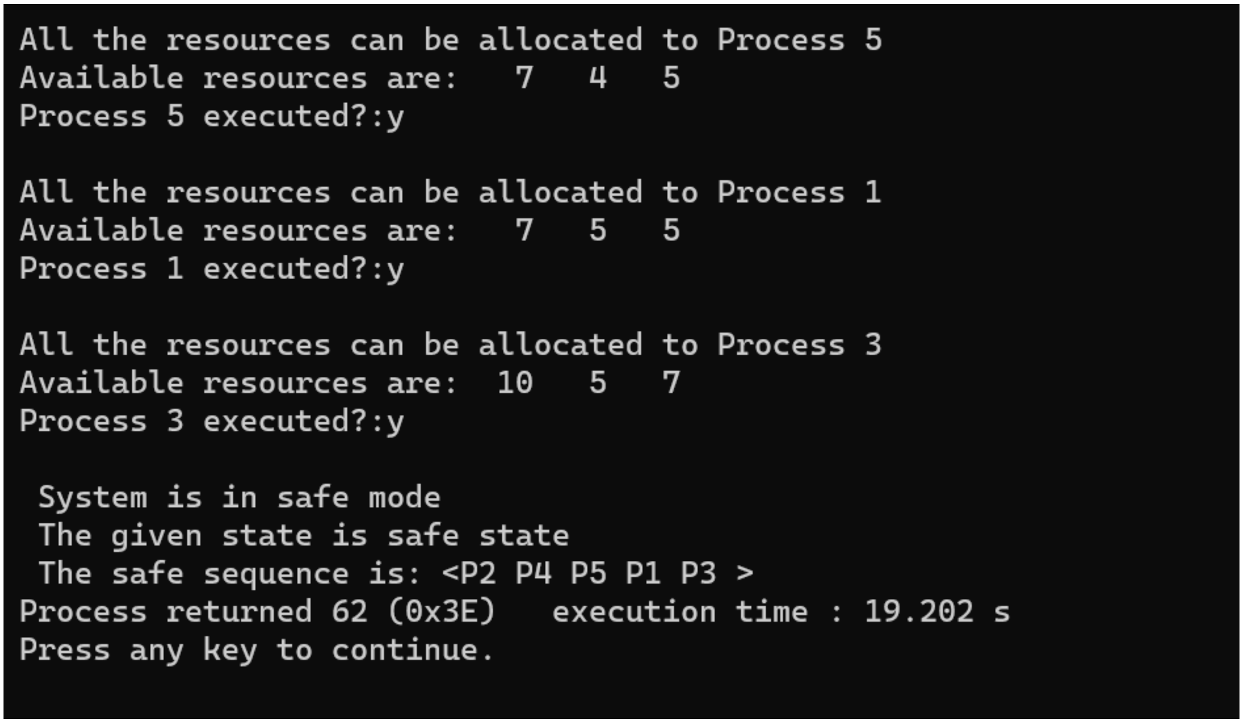
```
Enter the number of processes and recources:5 3
Enter the Maximum matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

Enter the allocation matrise :
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Resource vector : 3 3 2
```

```
Need Matrix :
7 4 3
1 2 2
6 0 0
0 1 1
4 3 1

All the resources can be allocated to Process 2
Available resources are : 5 3 2
Process 2 executed ? : y

All the resources can be allocated to Process 4
Available resources are : 7 4 3
Process 4 executed ? : y

All the resources can be allocated to Process 5
Available resources are : 7 4 5
Process 5 executed ? : y

All the resources cane be allocated to Process 1
Available resources are : 7 5 5
Process 1 executed ? : y

All the resources can be allocated to Process 3
Available resources are : 10 5 7
Process 3 executed ? : y

System is in safe mode
The given state is safe state
The safe sequence is : < P2 P4 P5 P1 P3 >
```

OUTPUT :



```
Enter the number of processes and resources:5 3
Enter the Maximum matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter the allocation matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Resource vector:3 3 2
Need Matrix:
7 4 3
1 2 2
6 0 0
0 1 1
4 3 1

All the resources can be allocated to Process 2
Available resources are:   5   3   2
Process 2 executed?:y

All the resources can be allocated to Process 4
Available resources are:   7   4   3
Process 4 executed?:y
```



```
All the resources can be allocated to Process 5
Available resources are:   7   4   5
Process 5 executed?:y

All the resources can be allocated to Process 1
Available resources are:   7   5   5
Process 1 executed?:y

All the resources can be allocated to Process 3
Available resources are:  10   5   7
Process 3 executed?:y

 System is in safe mode
 The given state is safe state
 The safe sequence is: <P2 P4 P5 P1 P3 >
Process returned 62 (0x3E)   execution time : 19.202 s
Press any key to continue.
```