VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

OPERATING SYSTEMS

Submitted by

AMISH B HARSOOR(1BM21CS017)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
June-2023 to September-2023

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "OPERATING SYSTEMS" carried out by **AMISH B HARSOOR(1BM21CS017),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the academic semester June-2023 to September-2023. The Lab report has been approved as it satisfies the academic requirements in respect of a OPERATING SYSTEMS **(22CS4PCOPS)** work prescribed for the said degree.

Basavaraj Jakkali	Dr. Jyothi S Nayak
Associate Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

Course Outcome

CO1	Apply the different concepts and functionalities of Operating System
CO2	Analyse various Operating system strategies and techniques
CO3	Demonstrate the different functionalities of Operating System.
CO4	Conduct practical experiments to implement the functionalities of Operating system.

1. Write a C program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time.

FCFS

SJF (pre-emptive & Non-pre-emptive)

Code:

```
#include <stdio.h>
int at[10], pt[10], ia, ip, n;
int tat[10], wt[10], it, iw, pos, j, i;
float atat = 0, awt = 0;
void fcfs()
{
   int t;
   printf("Enter number of processes: ");
   scanf("%d", &n);

   printf("Enter arrival times:\n");
   for (ia = 0; ia < n; ia++)
      scanf("%d", &at[ia]);

   printf("Enter process times:\n");
   for (ip = 0; ip < n; ip++)
      scanf("%d", &pt[ip]);

   if (at[0] == at[1])
   {
      t = pt[1];
      pt[1] = pt[0];
      pt[0] = t;
   }

   if (at[0] != 0)
      tat[0] = at[0];

   for (it = 0; it < n; it++)
      tat[it] = 0;
```

```
    int i = 0;
    for (it = 0; it < n; it++)
    {
        while (i <= it)
            tat[it] += pt[i++];
        i = 0;
    }

    for (it = 0; it < n; it++)
        tat[it] = tat[it] - at[it];

    for (ia = 0; ia < n; ia++)
        wt[ia] = tat[ia] - pt[ia];

    for (i = 0; i < n; i++)
    {
        atat += tat[i];
        awt += wt[i];
    }

    atat = atat / n;
    awt = awt / n;

    for (i = 0; i < n; i++)
    {
        printf("P%d\t%d\t%d\n", i, tat[i], wt[i]);
    }

    printf("Average TAT=%.2f\nAverage WT=%.2f\n", atat, awt);
}

void srtf()
{
    int rt[10], endTime, i, smallest;
    int remain = 0, time, sum_wait = 0, sum_turnaround = 0;
    printf("Enter no of Processes : ");
    scanf("%d",  &n);
    printf("Enter arrival times\n");
    for (i = 0; i < n; i++)
    {
```

```
        scanf("%d", &at[i]);
    }
    printf("Enter Process times \n");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &pt[i]);
        rt[i] = pt[i];
    }
    rt[9] = 9999;
    for (time = 0; remain != n; time++)
    {
        smallest = 9;
        for (i = 0; i < n; i++)
        {
            if (at[i] <= time && rt[i] < rt[smallest] && rt[i] > 0)
            {
                smallest = i;
            }
        }
        rt[smallest]--;
        if (rt[smallest] == 0)
        {
            remain++;
            endTime = time + 1;
            printf("\nP%d  %d  %d", smallest + 1, endTime - at[smallest], endTime - pt[smallest] -
at[smallest]);
            sum_wait += endTime - pt[smallest] - at[smallest];
            sum_turnaround += endTime - at[smallest];
        }
    }
    printf("\n\nAverage waiting time = %f\n", sum_wait * 1.0 / n);
    printf("Average Turnaround time = %f", sum_turnaround * 1.0 / n);
}

void sjf()
{
    int completed = 0;
    int currentTime = 0;
    int complete[n], ct[n];
```

```
printf("Enter number of processes: ");
scanf("%d", &n);

printf("Enter arrival times:\n");
for (int ia = 0; ia < n; ia++)
   scanf("%d", &at[ia]);

printf("Enter process times:\n");
for (int ip = 0; ip < n; ip++)
   scanf("%d", &pt[ip]);

for (int i = 0; i < n; i++)
{
   complete[i] = 0;
   ct[i] = 0;
}

while (completed != n)
{
   int shortest = -1;
   int min_bt = 9999;

   for (int i = 0; i < n; i++)
   {
      if (at[i] <= currentTime && complete[i] == 0)
      {
         if (pt[i] < min_bt)
         {
            min_bt = pt[i];
            shortest = i;
         }
         if (pt[i] == min_bt)
         {
            if (at[i] < at[shortest])
            {
               shortest = i;
            }
         }
      }
   }
```

```
        if (shortest == -1)
        {
            currentTime++;
        }
        else
        {
            ct[shortest] = currentTime + pt[shortest];
            tat[shortest] = ct[shortest] - at[shortest];
            wt[shortest] = tat[shortest] - pt[shortest];
            complete[shortest] = 1;
            completed++;
            currentTime = ct[shortest];
        }
    }
    for (int i = 0; i < n; i++)
    {
        atat += tat[i];
        awt += wt[i];
    }

    atat = atat / n;
    awt = awt / n;

    for (int i = 0; i < n; i++)
    {
        printf("P%d\t%d\t%d\n", i, tat[i], wt[i]);
    }

    printf("\nAverage TAT = %f\nAverage WT = %f\n", atat, awt);
}

void main()
{
    int op = 1, x;
    printf("1.FCFS \n2.SJF \n3.SRTF\n");
    scanf("%d", &x);
    switch (x)
    {
    case 1:
        fcfs();
```

```
        break;
    case 2:
        sjf();
        break;

    case 3:
        srtf();
        break;

    default:
        printf("Invalid option \n");
    }
}
```

Output:

```
PS D:\VS Code\OS> cd "d:\VS Code\OS\" ; if ($?) { gcc os.c -o os } ; if ($?) { .\os }
1.FCFS
2.SJF
3.SRTF
1
Enter number of processes: 3
Enter arrival times:
0 0 1
Enter process times:
8 4 1
P0      4       0
P1      12      4
P2      12      11
Average TAT=9.33
Average WT=5.00
```

```
PS D:\VS Code\OS> cd "d:\VS Code\OS\" ; if ($?) { gcc os.c -o os } ; if ($?) { .\os }
1.FCFS
2.SJF
3.SRTF
2
Enter number of processes: 3
Enter arrival times:
0 0 1
Enter process times:
8 4 1
P0      13      5
P1      4       0
P2      4       3

Average TAT = 7.000000
Average WT = 2.666667
PS D:\VS Code\OS>
```

```
PS D:\VS Code\OS> cd "d:\VS Code\OS\" ; if ($?) { gcc os.c -o os } ; if ($?) { .\os }
1.FCFS
2.SJF
3.SRTF
3
Enter no of Processes : 3
Enter arrival times
0 0 1
Enter Process times
8 4 1

P3  1  0
P2  5  1
P1  13  5

Average waiting time = 2.000000
Average Turnaround time = 6.333333
PS D:\VS Code\OS>
```

2. Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time.
Priority (pre-emptive & Non-pre-emptive)
Round Robin (Experiment with different quantum sizes for RR algorithm)

Code:

```
#include<stdio.h>

int at[10],t,pt[10],tat[10],wt[10],n,time=0,i,ready[10],pry[10],op=0, maxpr,x,p[10];
float atat=0,awt=0;

void main()
{
   printf("Enter number of processes \n");
   scanf("%d",&n);

   printf("Enter araival times: \n");
   for(i=0;i<n;i++)
   scanf("%d",&at[i]);

   printf("Enter process times: \n");
   for(i=0;i<n;i++)
   scanf("%d",&pt[i]);

   printf("Enter priority: \n");
   for(i=0;i<n;i++)
   scanf("%d",&pry[i]);

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

   for(i=0;i<n;i++)
   p[i]=pt[i];

   for(i=0;i<n;i++)
   time+=pt[i];
   t=n;
   while(t--)
```

```
{
   for(i=0;i<n;i++)
   if(op>=at[i])
   ready[i]=1;

   for(i=0;i<n;i++)
   if(pt[i]==0)
   pry[i]=0;


   //finding index of max priority
   maxpr=pry[0];
   for(i=0;i<n;i++)
   if(ready[i]==1)
   if(pry[i]>maxpr)
   maxpr=pry[i];

   for(i=0;i<n;i++)
   if(maxpr==pry[i])
   x=i;

   //printing chart
   printf("%d p%d ",op,(x+1));


   op=op+pt[x];
   tat[x]=op;
   ready[x]=0;
   pry[x]=0;
}
printf("%d",op);

//finding avgtat and avg wt
for(i=0;i<n;i++)
{
   tat[i]=tat[i]-at[i];
}

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

```
    atat+=tat[i];
    wt[i]=tat[i]-pt[i];
  }
  for(i=0;i<n;i++)
  awt+=wt[i];

  awt=awt/n;
  atat=atat/n;

  //printing final values
  printf("\n");
  for(i=0;i<n;i++)
  printf("P%d %d %d \n",(i+1),tat[i],wt[i]);
  printf("ATAT=%f \nAWT=%f ",atat,awt);
}
```

Output:


```
#include<stdio.h>

  int tq, at[10], pt[10], p[10], time=0, op=0, i,j ,n, ready[10],q[100];
  int r=-1,f=0,tat[10],wt[10],z,fg,y=9999,ch;
  float atat,awt;

int rr(int x)
{
  if(pt[x]>tq)
  {
    pt[x]-=tq;
    op+=tq;
  }
  else
```

```
    {
        op+=pt[x];
        pt[x]=0;
        tat[x]=op;
        ready[x]=0;
    }
    return x;
}

void main()
{
    printf("Enter number or processes \n");
    scanf("%d",&n);

    printf("Enter araival times: \n");
    for(i=0;i<n;i++)
    scanf("%d",&at[i]);

    printf("Enter process times: \n");
    for(i=0;i<n;i++)
    scanf("%d",&pt[i]);

    printf("Enter TQ \n");
    scanf("%d",&tq);

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

    for(i=0;i<n;i++)
    q[i]=9999;

    for(i=0;i<n;i++)
    p[i]=pt[i];

    for(i=0;i<n;i++)
    time+=pt[i];


    for(i=0;i<n;i++)
        if(op>=at[i])
```

```
    ready[i]=1;

for(i=0;i<n;i++)
    if(ready[i]==1)
    {
        q[++r]=i;
    }

while(op!=time)
{
    printf("%d ",op);
    if(z==y)
    q[++f];
    y=z;

    ch=q[f];
    if(pt[ch]!=0)
    {
    z=rr(q[f]);

    printf("P%d ",(z+1));

    for(i=0;i<n;i++)
    {
      if(op>=at[i] && pt[i]!=0)
      {
       fg=0;
       j=f;
       while(j<=r)
       {
          if(i==q[j])
          fg=1;
          j++;
       }
       if(fg==0)
       {
          q[++r]=i;
       }
      }
    }
```

```
        if(pt[z]!=0)
        q[++r]=z;
        }
        f++;
    }

    printf("%d ",op);

    for(i=0;i<n;i++)
    {
        tat[i]=tat[i]-at[i];
        wt[i]=tat[i]-p[i];
        atat+=tat[i];
        awt+=wt[i];
    }
    atat=atat/n;
    awt=awt/n;

    printf("\n");
    for(i=0;i<n;i++)
    printf("P%d %d %d \n",(i+1),tat[i],wt[i]);
    printf("ATAT=%f \nAWT=%f ",atat,awt);
}
```

Output:

3. Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories ± system processes and user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for the processes in each queue.

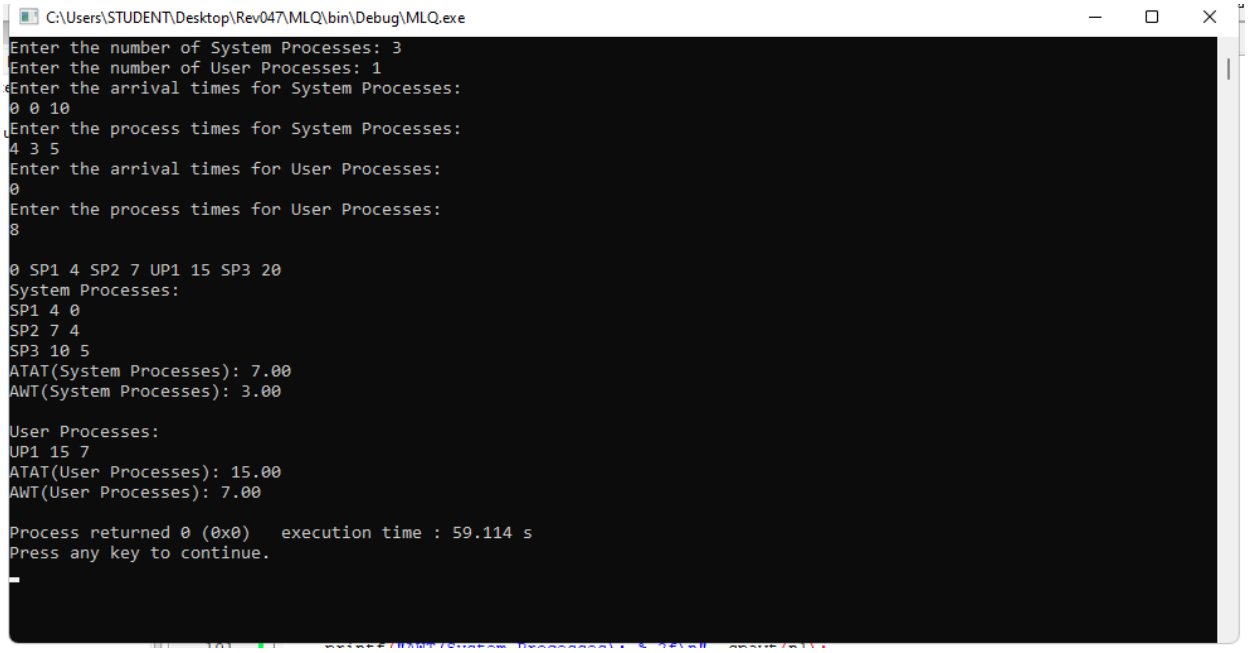
Code:

```
#include <stdio.h>

int spat[10], upat[10], i, n1, n2, p1[10], p2[10];
int sppt[10], uppt[10], time = 0, op = 0, y, z, pt;
int sptat[10], uptat[10];
int spwt[10], upwt[10];
float spatat = 0, spawt = 0;
float upatat = 0, upawt = 0;

void process(int x, int isSystem) {
   if (isSystem) {
      op +=  sppt[x];
      sptat[x] = op - spat[x];
      sppt[x] = 0;
      spwt[x] = sptat[x] - p1[x];
      spatat += sptat[x];
      spawt += spwt[x];
   } else {
      op +=  uppt[x];
      uptat[x] = op - upat[x];
      uppt[x] = 0;
      upwt[x] = uptat[x] - p2[x];
      upatat += uptat[x];
      upawt += upwt[x];
   }
}

int main() {
   printf("Enter the number of System Processes: ");
   scanf("%d", &n1);

   printf("Enter the number of User Processes: ");
   scanf("%d", &n2);
```

```
printf("Enter the arrival times for System Processes:\n");
for (i = 0; i < n1; i++)
   scanf("%d", &spat[i]);

printf("Enter the process times for System Processes:\n");
for (i = 0; i < n1; i++)
   scanf("%d", &sppt[i]);

printf("Enter the arrival times for User Processes:\n");
for (i = 0; i < n2; i++)
   scanf("%d", &upat[i]);

printf("Enter the process times for User Processes:\n");
for (i = 0; i < n2; i++)
   scanf("%d", &uppt[i]);

for (i = 0; i < n1; i++)
   time += sppt[i];

for (i = 0; i < n2; i++)
   time += uppt[i];

for (i = 0; i < n1; i++)
   p1[i] = sppt[i];

for (i = 0; i < n2; i++)
   p2[i] = uppt[i];

printf("\n");
while (op < time) {
   y = -1;
   z = -1;
   for (i = 0; i < n1; i++) {
      if (op >= spat[i] && sppt[i] != 0) {
         y = i;
         break;
      }
   }
   for (i = 0; i < n2; i++) {
```

```
            if (op >= upat[i] && uppt[i] != 0) {
                z = i;
                break;
            }
        }
        if (y != -1) {
            printf("%d SP%d ", op, y + 1);
            process(y, 1);
        } else if (z != -1) {
            printf("%d UP%d ", op, z + 1);
            process(z, 0);
        } else {
            op++;
        }
    }
    printf("%d ",op);
    printf("\n");
    printf("System Processes:\n");
    for (i = 0; i < n1; i++)
        printf("SP%d %d %d\n", i + 1, sptat[i],spwt[i]);
    printf("ATAT(System Processes): %.2f\n", spatat / n1);
    printf("AWT(System Processes): %.2f\n", spawt/n1);
    printf("\n");
    printf("User Processes:\n");
    for (i = 0; i < n2; i++)
        printf("UP%d %d %d\n", i + 1, uptat[i], upwt[i]);
    printf("ATAT(User Processes): %.2f\n", upatat / n2);
    printf("AWT(User Processes): %.2f\n", upawt / n2);
    return 0;
}
```

Output:


C:\Users\STUDENT\Desktop\Rev047\MLQ\bin\Debug\MLQ.exe
Enter the number of System Processes: 3
Enter the number of User Processes: 1
Enter the arrival times for System Processes:
0 0 10
Enter the process times for System Processes:
4 3 5
Enter the arrival times for User Processes:
0
Enter the process times for User Processes:
8
0 SP1 4 SP2 7 UP1 15 SP3 20
System Processes:
SP1 4 0
SP2 7 4
SP3 10 5
ATAT(System Processes): 7.00
AWT(System Processes): 3.00
User Processes:
UP1 15 7
ATAT(User Processes): 15.00
AWT(User Processes): 7.00
Process returned 0 (0x0) execution time : 59.114 s
Press any key to continue.

4. Write a C program to simulate Real-Time CPU Scheduling algorithms:
a) Rate- Monotonic
b) Earliest-deadline First

Code:

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


int et[10], i, n, dl[10], p[10], ready[10], flag = 1;

int lcm(int a, int b) {
    int max = (a > b) ? a : b;
    while (1) {
        if (max % a == 0 && max % b == 0)
            return max;
        max++;
    }
}

int lcmArray(int arr[], int n) {
    int result = arr[0];
    for (int i = 1; i < n; i++) {
        result = lcm(result, arr[i]);
    }
    return result;
}

void mono() {
    int time = lcmArray(dl, n);
    int op = 0, pr = 0, pre = pr;



    while (op <= time) {
        for (i = 0; i < n; i++) {
            if (op % dl[i] == 0) {
                ready[i] = 1;
```

```
    }
}

flag = 0;
for (i = 0; i < n; i++) {
    if (ready[i] == 1) {
        flag = 1;
        break;
    }
}

if (flag == 0) {
    pr = -1;
} else {
    pr = -1;
    for (i = 0; i < n; i++) {
        if (ready[i] == 1) {
            if (pr == -1 || dl[i] < dl[pr]) {
                pr = i;
            }
        }
    }
}


    if (pr != pre) {
        if (pr == -1) {
            printf("%d Idle ",op);
        } else {
            printf("%d P%d ",op, pr + 1);
        }
    }

op++;
if (pr != -1) {
    p[pr] = p[pr] - 1;
    if (p[pr] == 0) {
        p[pr] = et[pr];
        ready[pr] = 0;
    }
```

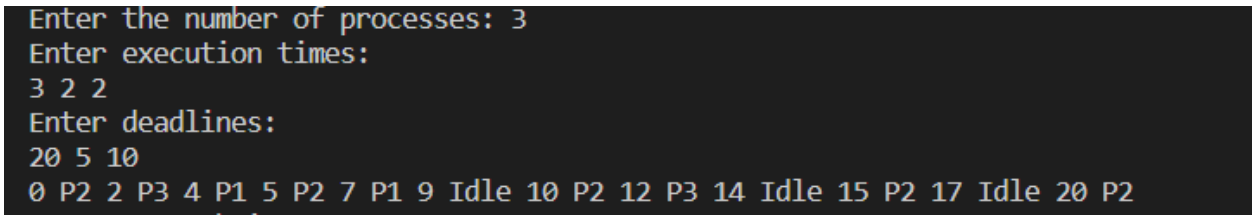
```
        }

        pre = pr;
    }

    printf("\n");
}



void edf() {
    int time = lcmArray(dl, n);
    int op = 0, pr = 0, pre = -1;
    int flag, i;

    while (op <= time) {
        for (i = 0; i < n; i++) {
            if (op % dl[i] == 0) {
                ready[i] = 1;
            }
        }

        flag = 0;
        for (i = 0; i < n; i++) {
            if (ready[i] == 1) {
                flag = 1;
                break;
            }
        }

        if (flag == 0) {
            pr = -1;
        } else {
            pr = -1;
            for (i = 0; i < n; i++) {
                if (ready[i] == 1) {
                    if (pr == -1 || p[i] < p[pr]) {
                        pr = i;
                    }
                }
```

```
            }
        }

        if (pr != pre) {
            if (pr == -1) {
                printf("%d Idle ", op);
            } else {
                printf("%d P%d ", op, pr + 1);
            }
        }

        op++;

        if (pr != -1) {
            p[pr] = p[pr] - 1;
            if (p[pr] == 0) {
                p[pr] = et[pr];
                ready[pr] = 0;
            }
        }

        pre = pr;
    }

    printf("\n");
}



int main() {
    int ch, k = 1;
    while (k) {
        printf("Enter your choice: \n1. Monotonic \n2. EDF \n4. Exit\n");
        scanf("%d", &ch);
        if(ch==4)
        exit(0);
        printf("Enter the number of processes: ");
        scanf("%d", &n);
        printf("Enter execution times: \n");
        for (i = 0; i < n; i++)
```

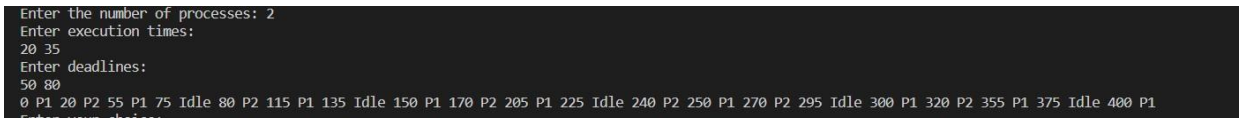
```
        scanf("%d", &et[i]);

    printf("Enter deadlines: \n");
    for (i = 0; i < n; i++)
        scanf("%d", &dl[i]);

    for (i = 0; i < n; i++)
        p[i] = et[i];

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

    switch (ch) {
        case 1:
            mono();
            break;

        case 2:
            edf();
            break;

        case 3:
            prop();
            break;

            default:
            printf("Invalid choice.\n");
    }
  }

  return 0;
}
```

Output:

Rate Monotonic:



Earliest Deadline First:



5. Write a C program to simulate producer-consumer problem using Semaphores.

Code:

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

int mutex=1,full=0,empty=3,x=0;

int main()
{
   int n;
   void producer();
   void consumer();
   int wait(int);
   int signal(int);
   printf("\n1.Producer\n2.Consumer\n3.Exit");
   while(1)
   {
      printf("\nEnter your choice:");
      scanf("%d",&n);
      switch(n)
      {
         case 1:    if((mutex==1)&&(empty!=0))
                       producer();
                    else
                       printf("Buffer is full!!");
                    break;
         case 2:    if((mutex==1)&&(full!=0))
                       consumer();
                    else
                       printf("Buffer is empty!!");
                    break;
         case 3:
                    exit(0);
                    break;
      }
   }
```

```
    return 0;
}

int wait(int s)
{
    return (--s);
}

int signal(int s)
{
    return(++s);
}

void producer()
{
    mutex=wait(mutex);
    full=signal(full);
    empty=wait(empty);
    x++;
    printf("\nProducer produces the item %d",x);
    mutex=signal(mutex);
}

void consumer()
{
    mutex=wait(mutex);
    full=wait(full);
    empty=signal(empty);
    printf("\nConsumer consumes item %d",x);
    x--;
    mutex=signal(mutex);
}
```

Output:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PS D:\VS Code> cd "d:\VS Code\OS\" ; if ($?) { gcc podcon.c -o podcon } ; if ($?) { .\podcon }
1.Producer
2.Consumer
3.Exit
Enter your choice:1
Producer produces the item 1
Enter your choice:1
Producer produces the item 2
Enter your choice:2
Consumer consumes item 2
Enter your choice:2
Consumer consumes item 1
Enter your choice:2
Buffer is empty!!
Enter your choice:3
PS D:\VS Code\OS>
Code
Code
Code

6. Write a C program to simulate the concept of Dining-Philosophers problem.

Code:

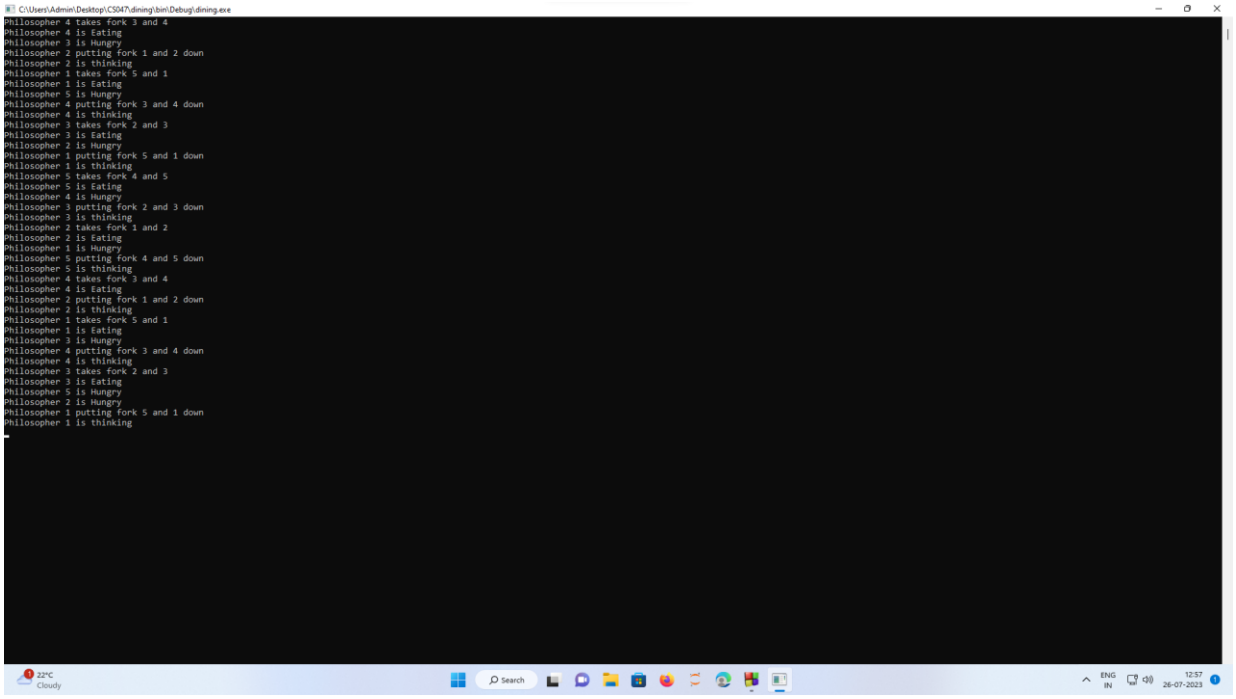
```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>

#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 that 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]) has no effect
      // during takefork
      // used to wake up hungry philosophers
```

```
        // during putfork
        sem_post(&S[phnum]);
    }
}

// take up chopsticks
void take_fork(int phnum)
{

    sem_wait(&mutex);

    // state that hungry
    state[phnum] = HUNGRY;

    printf("Philosopher %d is Hungry\n", phnum + 1);

    // eat if neighbours are not eating
    test(phnum);

    sem_post(&mutex);

    // if unable to eat wait to be signalled
    sem_wait(&S[phnum]);

    sleep(1);
}

// put down chopsticks
void put_fork(int phnum)
{

    sem_wait(&mutex);

    // state that thinking
    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];

    // initialize the semaphores
    sem_init(&mutex, 0, 1);

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

        sem_init(&S[i], 0, 0);

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

        // create philosopher processes
        pthread_create(&thread_id[i], NULL,
```

```
                philosopher, &phil[i]);

        printf("Philosopher %d is thinking\n", i + 1);
    }

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

        pthread_join(thread_id[i], NULL);
}
```

Output:



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

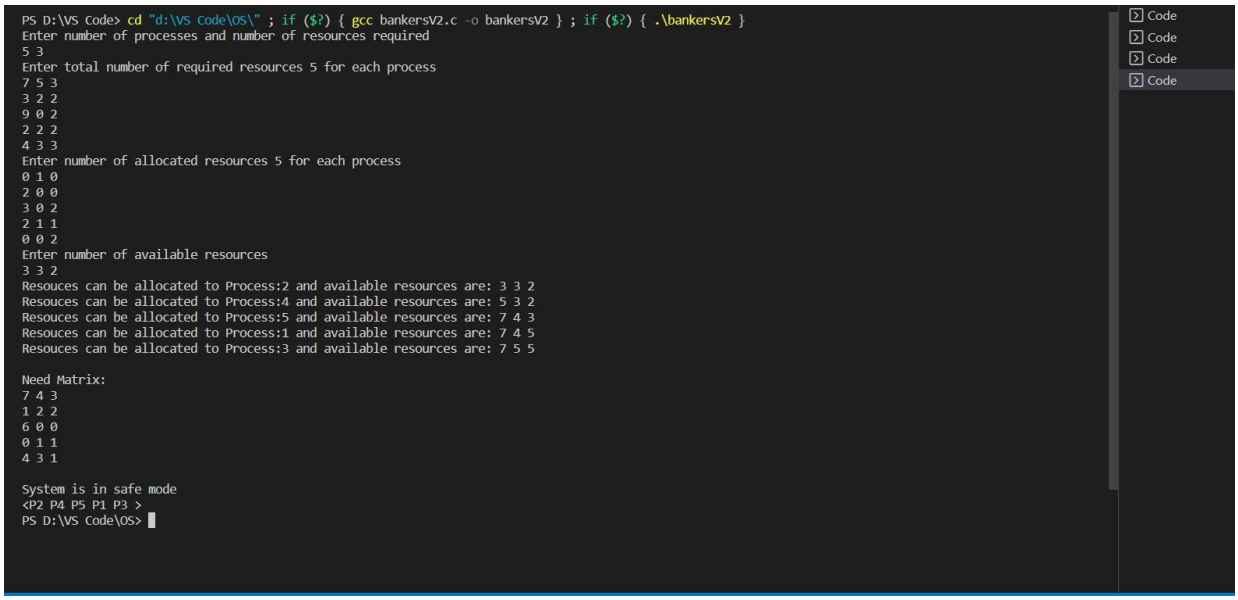
Code:

```
#include <stdio.h>

int main() {
   int n, m, all[10][10], req[10][10], ava[10], need[10][10];
   int i, j, k, flag[10], prev[10], c, count = 0,array[10],z=0;

   printf("Enter number of processes and number of resources required \n");
   scanf("%d %d", &n, &m);

   printf("Enter total number of required resources %d for each process\n", n);
   for (i = 0; i < n; i++)
      for (j = 0; j < m; j++)
         scanf("%d", &req[i][j]);

   printf("Enter number of allocated resources %d for each process\n", n);
   for (i = 0; i < n; i++)
      for (j = 0; j < m; j++)
         scanf("%d", &all[i][j]);

   printf("Enter number of available resources \n");
   for (i = 0; i < m; i++)
      scanf("%d", &ava[i]);

   for (i = 0; i < n; i++)
      for (j = 0; j < m; j++)
         need[i][j] = req[i][j] - all[i][j];

   for (i = 0; i < n; i++)
      flag[i] = 1;

   k = 1;

   while (k) {
      k = 0; // Reset the value of k for each iteration of the loop

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

```
        if (flag[i]) {
            c = 0;
            for (j = 0; j < m; j++) {
                if (need[i][j] <= ava[j]) {
                    c++;
                }
            }
            if (c == m) {
                    array[z++]=i;
                printf("Resouces can be allocated to Process:%d and available resources are: ", (i +
1));
                for (j = 0; j < m; j++) {
                    printf("%d ", ava[j]);
                }
                printf("\n");

                for (j = 0; j < m; j++) {
                    ava[j] += all[i][j];
                    all[i][j] = 0;
                }

                flag[i] = 0;
                count++;
            }
        }
    }

    // Check if the current state is different from the previous state
    for (i = 0; i < n; i++) {
        if (flag[i] != prev[i]) {
            k = 1;
            break;
        }
    }

    for (i = 0; i < n; i++) {
        prev[i] = flag[i];
    }
  }
```

```
    printf("\nNeed Matrix:\n");
    for (i = 0; i < n; i++) //printing need matrix
    {
        for (j = 0; j < m; j++)
            printf("%d ",need[i][j]);
        printf("\n");
    }


    if (count == n) {
        printf("\nSystem is in safe mode \n<");
        for(i=0;i<n;i++)
        printf("P%d ",(array[i]+1));
        printf(">\n");
    } else {
        printf("\nSystem is not in safe mode deadlock occurred \n");
    }
    return 0;
}
```

Output:



```
Enter number of processes and number of resources required
5 3
Enter total number of required resources 5 for each process
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter number of allocated resources 5 for each process
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter number of available resources
2 2 2
Resouces can be allocated to Process:2 and available resources are: 2 2 2
Resouces can be allocated to Process:4 and available resources are: 4 2 2
Resouces can be allocated to Process:5 and available resources are: 6 3 3
Resouces can be allocated to Process:3 and available resources are: 6 3 5

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

System is not in safe mode deadlock occurred
```

8. Write a C program to simulate deadlock detection

Code:

```
#include <stdio.h>

int main() {
   int n, m, all[10][10], req[10][10], ava[10], need[10][10];
   int i, j, k, flag[10], prev[10], c, count = 0;

   printf("Enter number of processes and number of resources required \n");
   scanf("%d %d", &n, &m);

   printf("Enter total number of required resources %d for each process\n", n);
   for (i = 0; i < n; i++)
      for (j = 0; j < m; j++)
         scanf("%d", &req[i][j]);

   printf("Enter number of allocated resources %d for each process\n", n);
   for (i = 0; i < n; i++)
      for (j = 0; j < m; j++)
         scanf("%d", &all[i][j]);

   printf("Enter number of available resources \n");
   for (i = 0; i < m; i++)
      scanf("%d", &ava[i]);

   for (i = 0; i < n; i++)
      for (j = 0; j < m; j++)
         need[i][j] = req[i][j] - all[i][j];

   for (i = 0; i < n; i++)
      flag[i] = 1;

   k = 1;
   while (k) {
      k = 0; // Reset the value of k for each iteration of the loop
      for (i = 0; i < n; i++) {
         if (flag[i]) {
            c = 0;
            for (j = 0; j < m; j++) {
```

```
                if (need[i][j] <= ava[j]) {
                    c++;
                }
            }
            if (c == m) {
                for (j = 0; j < m; j++) {
                }
                for (j = 0; j < m; j++) {
                    ava[j] += all[i][j];
                    all[i][j] = 0;
                }
                flag[i] = 0;
                count++;
            }
        }
    }
    // Check if the current state is different from the previous state
    for (i = 0; i < n; i++) {
        if (flag[i] != prev[i]) {
            k = 1;
            break;
        }
    }
    for (i = 0; i < n; i++) {
        prev[i] = flag[i];
    }
  }
  if (count == n) {
    printf("\nNo deadlock");
  } else {
    printf("\nDeadlock occurred \n");
  }
  return 0;
}
```

Output:

```
Enter number of processes and number of resources required
3
3
Enter total number of required resources 3 for each process
6 2 1
3 5 2
1 1 2
Enter number of allocated resources 3 for each process
4 0 1
2 3 0
0 0 1
Enter number of available resources
2 2 2

No deadlock
```

```
Enter number of processes and number of resources required
3 3
Enter total number of required resources 3 for each process
7 5 2
4 4 3
3 3 3
Enter number of allocated resources 3 for each process
2 0 0
1 0 0
1 1 1
Enter number of available resources
2 2 2

Deadlock occurred
```

9. Write a C program to simulate the following contiguous memory allocation techniques
a) Worst-fit
b) Best-fit
c) First-fit

Code:

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

#define max 25

void main()
{
   int frag[max], b[max], f[max], i, j, nb, nf, temp;
   int bf[max], ff[max];

   printf("\n\tMemory Management Scheme - First Fit");
   printf("\nEnter the number of blocks:");
   scanf("%d", &nb);
   printf("Enter the number of files:");
   scanf("%d", &nf);

   printf("\nEnter the size of the blocks:\n");
   for (i = 1; i <= nb; i++)
   {
      printf("Block %d:", i);
      scanf("%d", &b[i]);
   }

   printf("Enter the size of the files:\n");
   for (i = 1; i <= nf; i++)
   {
      printf("File %d:", i);
      scanf("%d", &f[i]);
   }

   for (i = 1; i <= nf; i++)
   {
      temp = -1; // Reset temp to -1 for each new file
```

```
        for (j = 1; j <= nb; j++)
        {
            if (bf[j] != 1)
            {
                if (b[j] >= f[i])
                {
                    ff[i] = j;
                    temp = b[j] - f[i];
                    break;
                }
            }
        }
        frag[i] = temp;
        if (temp != -1)
        {
            bf[ff[i]] = 1;
        }
    }

    printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment");
    for (i = 1; i <= nf; i++)
    {
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
    }

    getch();
}
```

Output:

Code:

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

#define max 25

void main()
{
   int frag[max], b[max], f[max], i, j, nb, nf, temp, lowest = 10000;
   static int bf[max], ff[max];

   printf("\nEnter the number of blocks:");
   scanf("%d", &nb);
   printf("Enter the number of files:");
   scanf("%d", &nf);

   printf("\nEnter the size of the blocks:\n");
   for (i = 1; i <= nb; i++)
   {
      printf("Block %d:", i);
      scanf("%d", &b[i]);
   }

   printf("Enter the size of the files:\n");
   for (i = 1; i <= nf; i++)
   {
      printf("File %d:", i);
      scanf("%d", &f[i]);
   }

   for (i = 1; i <= nf; i++)
   {
      lowest = 10000; // Reset lowest to a high value for each new file
      for (j = 1; j <= nb; j++)
      {
         if (bf[j] != 1)
         {
            temp = b[j] - f[i];
            if (temp >= 0 && lowest > temp)
            {
```

```
                ff[i] = j;
                lowest = temp;
            }
        }
    }
    frag[i] = lowest;
    bf[ff[i]] = 1;
}

printf("\nFile No\tFile Size\tBlock No\tBlock Size\tFragment");
for (i = 1; i <= nf && ff[i] != 0; i++)
{
    printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
}

getch();
}
```

Output:

```
PS D:\VS Code\OS> cd "d:\VS Code\OS\" ; if ($?) { gcc fit.c -o fit } ; if ($?) { .\fit }

Enter the number of blocks:3
Enter the number of files:2

Enter the size of the blocks:
Block 1:5
Block 2:2
Block 3:7
Enter the size of the files:
File 1:1
File 2:4

File No File Size       Block No        Block Size      Fragment
1               1               2               2               1
2               4               1               5               1
```

Code:

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

#define max 25

void main()
{
   int frag[max], b[max], f[max], i, j, nb, nf, temp, highest = 0;
   int bf[max], ff[max]; // Initialized these arrays to 0
   printf("\n\tMemory Management Scheme - Worst Fit");
   printf("\nEnter the number of blocks:");
   scanf("%d", &nb);
   printf("Enter the number of files:");
   scanf("%d", &nf);
   printf("\nEnter the size of the blocks:\n");
   for (i = 1; i <= nb; i++)
   {
      printf("Block %d:", i);
      scanf("%d", &b[i]);
   }
   printf("Enter the size of the files:\n");
   for (i = 1; i <= nf; i++)
   {
      printf("File %d:", i);
      scanf("%d", &f[i]);
   }

   for (i = 1; i <= nf; i++)
   {
      highest = 0; // Reset highest to 0 for each new file
      for (j = 1; j <= nb; j++)
      {
         if (bf[j] != 1) // If bf[j] is not allocated
         {
            temp = b[j] - f[i];
            if (temp >= 0)
            {
               if (highest < temp)
               {
```

```
                ff[i] = j;
                highest = temp;
            }
        }
    }
    frag[i] = highest;
    bf[ff[i]] = 1;
}

printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragement");
for (i = 1; i <= nf; i++)
{
    printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
}

getch();
}
```

Output:

10. Write a C program to simulate page replacement algorithms
a) FIFO
b) LRU
c) Optimal

Code:

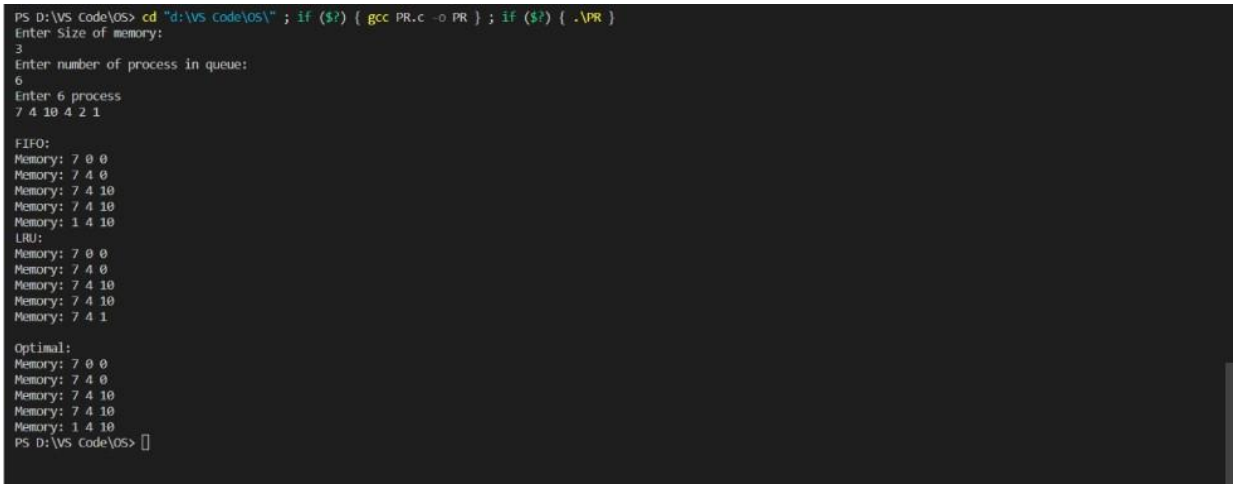
```
#include<stdio.h>

void main()
{
   int mem[20],process[20],n,m,i,j,k,c,z,a,distance=0,b;
   printf("Enter Size of memory:\n");
   scanf("%d",&n);
    for(i=0;i<n;i++)
        mem[i]=0;
   printf("Enter number of process in queue:\n");
   scanf("%d",&m);
   printf("Enter %d process \n",m);
   for(i=0;i<m;i++)
      scanf("%d",&process[i]);
   j=0;
   i=0;
   printf("\nFIFO:");
   while(j!=m)
   {
      k=0;
      c=0;
      while(k!=n)
      {
         c++;
         if(mem[k]==process[j])
         {
            j++;
            break;
         }
         k++;
      }
      if(c==n)
      {
         mem[i]=process[j];
```

```
        i=(i+1)%n;
    }
    printf("\nMemory: ");
    for(z=0;z<n;z++)
        printf("%d ",mem[z]);
    j++;
}
printf("\nLRU:");
for(i=0;i<n;i++)
    mem[i]=0;
i=0;
j=0;
while(j!=m)
{
    k=0;
    c=0;
    while(k!=n)
    {
        c++;
        if(mem[k]==process[j])
        {
            j++;
            break;
        }
        k++;
    }
    if(c==n)
    {
        distance=0;
        for(a=0;a<n;a++)
        {
            b=99;
            z=j;
            while(z>=0)
            {
                if((j-z)>distance)
                if(mem[a]==process[z])
                {
                    distance=(z-j);
                    b=z;
```

```
            }
            z--;
        }
    }
    if(b==99)
    b=i;
    mem[b]=process[j];
    i=(i+1)%n;
  }
  printf("\nMemory: ");
  for(z=0;z<n;z++)
    printf("%d ",mem[z]);
  j++;
}
printf("\n\nOptimal:");
for(i=0;i<n;i++)
  mem[i]=0;
i=0;
j=0;
while(j!=m)
{
  k=0;
  c=0;
  while(k!=n)
  {
    c++;
    if(mem[k]==process[j])
    {
      j++;
      break;
    }
    k++;
  }
  if(c==n)
  {
    distance=0;
    for(a=0;a<n;a++)
    {
      b=99;
      z=j;
```

```
            while(z!=m)
            {
               if((z-j)>distance)
               if(mem[a]==process[z])
               {
                  distance=(z-j);
                  b=z;
               }
               z++;
            }
         }
         if(b==99)
         b=i;
         mem[b]=process[j];
         i=(i+1)%n;
      }
      printf("\nMemory: ");
      for(z=0;z<n;z++)
         printf("%d ",mem[z]);
      j++;
   }
}
```

Output:



11. Write a C program to simulate disk scheduling algorithms
a) FCFS
b) SCAN
c) C-SCAN

Code:

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

int disks;
void quicksort(int number[25], int first, int last)
{
   int i, j, pivot, temp;
   if (first < last)
   {
      pivot = first;
      i = first;
      j = last;
      while (i < j)
      {
         while (number[i] <= number[pivot] && i < last)
            i++;
         while (number[j] > number[pivot])
            j--;
         if (i < j)
         {
            temp = number[i];
            number[i] = number[j];
            number[j] = temp;
         }
      }
      temp = number[pivot];
      number[pivot] = number[j];
      number[j] = temp;
      quicksort(number, first, j - 1);
      quicksort(number, j + 1, last);
   }
}
void fcfs(int arr[],int src, int n)
```

```
{
   int sseq[20],i;
   sseq[0]=abs(arr[0]-src);
   for(i=1;i<n;i++)
   sseq[i]=abs(arr[i]-arr[i-1]);
   int sum=0;
   for(i=0;i<n;i++)
   sum+=sseq[i];
   printf("\nFCFS \nTotal seek sequenece: %d \nSeek Sequence: \n",sum);
   for(i=0;i<n;i++)
   printf("%d ",sseq[i]);
   printf("\n");
}
void cscan(int arr[], int src, int n)
{
   int i,sum=0,j,sseq[20];
   quicksort(arr, 0, n-1);
   int index;
   for (index = 0; index < n; index++) {
      if (arr[index] == src) {
         break;
      }
   }
   i=index+1;
   j=0;
   while(i<=n)
   {
      sseq[j]=abs(arr[i]-arr[i-1]);
      i++;
      j++;
   }
   sseq[j++]=abs(disks-arr[i-1]);
   i=0;
   sseq[j++]=abs(disks);
   while(i<index)
   {
      sseq[j++]=abs(arr[i]-arr[i-1]);
      i++;
   }
   for(i=0;i<(n+2);i++)
```

```
    sum+=sseq[i];
    printf("\nC-SCAN \nTotal seek sequenece: %d \nSeek Sequence: \n",sum);
    for(i=0;i<n+2;i++)
    printf("%d ",sseq[i]);
    printf("\n");
}
void scan(int arr[], int src, int n)
{
    int i,sum=0,j,sseq[20];
    quicksort(arr, 0, n-1);
    int index;
    for (index = 0; index < n; index++) {
        if (arr[index] == src) {
            break;
        }
    }
    i=index-1;
    j=0;
    while(i>=0)
    {
        sseq[j]=abs(arr[i]-arr[i+1]);
        i--;
        j++;
    }
    i=index+1;
    sseq[j++]=abs(arr[i++]-arr[0]);
    while(i<=n)
    {
        sseq[j++]=abs(arr[i]-arr[i-1]);
        i++;
    }
    for(i=0;i<n;i++)
    sum+=sseq[i];
    printf("\nSCAN \nTotal seek sequenece: %d \nSeek Sequence: \n",sum);
    for(i=0;i<n;i++)
    printf("%d ",sseq[i]);
    printf("\n");
}
void main()
{
```

```
    int source, arr[20],i,n,copy[20];
    printf("Enter numebr of disks: ");
    scanf("%d",&n);
    printf("\nEnter %d values: ",n);
    for(i=0;i<n;i++)
    scanf("%d",&arr[i]);
    printf("\nEnter source position: ");
    scanf("%d",&source);
    printf("\nEnter number disks: ");
    scanf("%d",&disks);
    for(i=0;i<n;i++)
    copy[i]=arr[i];
    arr[n]=source;
    copy[n]=arr[n];
    fcfs(copy , source , n);
    scan(copy , source , n);
    cscan(arr , source , n);
}
```

Output: