Program (cp_ls)

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

void main(int argc, char *arg[])
{
    if (argc == 1)
    {
        DIR *p;
        struct dirent *d;
        p = opendir("./");
        if (p == NULL)
        {
            printf("No files found");
        }
        else
        {
            while (d = readdir(p))
            {
                printf("%s\n", d->d_name);
            }
        }
    }
    else
    {
        FILE *f1, *f2;
        char ch;
        f1 = fopen(arg[1], "r");
        f2 = fopen(arg[2], "w");

        while ((ch = fgetc(f1)) != EOF)
        {
            fputc(ch, f2);
        }

        printf("Contents copied successfully!\n");
        fclose(f1);
        fclose(f2);
    }
}
```

Output (cp_ls)

```
./cp_ls sample.txt new.txt
Contents copied successfully!

./cp_ls
ls
ls.c
cp_ls.c
prime.sh
cp.c
cp
fork
grep.c
fibanocci.sh
fork.c
grep
```

Program (grep)

```
#include <stdio.h>
#include <string.h>
#include <dirent.h>
void main(int argc,char *args[])
{
    char temp[200];
    FILE * fp;
    fp = fopen(args[2],"r");
    while(!feof(fp))
    {
        fgets(temp,sizeof(fp),fp);
        if(strstr(temp,args[1]))
        {
            printf("%s",temp);
        }
    }
    printf("\n");
    fclose(fp);
}
```

Output (GREP)

```
./grep "a" "sample.txt"
area
absolute
```

Program (fibonacci)

```
echo "enter number"
read n
a=0
b=1
c=0
i=1
while [ $i -le $n ]
do
    echo $a
    c=$(($a+$b))
    a=$b
    b=$c
    i=$(($i+1))
done
```

Output

```
Enter the number
10
0
1
1
2
3
5
8
13
21
34
```

Program (prime)

```
echo "Enter the number"
read n
i=2
while [ $i -le  $(($n/2)) ]
do
    if test `expr $n % $i` -eq 0
    then
        echo "$n is not a prime number"
        exit
    fi
    i=$(($i+1))
done
echo "$n is a Prime number"
```

Output

```
Enter the number
10
10 is not a prime number

Enter the number
7
7 is a prime number
```

Program (fork)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
int main()
{
    int pid;
    pid = fork();
    if(pid<0)
    {
        printf("fork failed");
    }
    else if(pid==0)
    {
        printf("inside child process ! \n");
        printf("Successfully forked process\n");
        printf("PID: %d",getpid());
        exit(0);
    }
    else
    {
        printf("Process id is - %d\n",getpid());
        exit(0);
    }
}
```

Output

```
Process id is - 14618
inside child process !
Successfully forked process
PID: 14619
```

Program (FCFS)

```
#include <stdio.h> //
int main()
{
    int n, i;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    int bt[n], at[n], wt[n], tat[n], ct[n];
    printf("Enter the arrival times of n processes: ");
    for (i = 0; i < n; i++)
    {
        scanf("%d", at + i);
    }
    printf("Enter the burst times of n processes: ");
    int sum = 0;
    for (i = 0; i < n; i++)
    {
        scanf("%d", bt + i);
        sum += bt[i];
        ct[i] = sum;
    }
    int totalTAT = 0, totalWT = 0;
    for (i = 0; i < n; i++)
    {
        tat[i] = ct[i] - at[i];
        totalTAT += tat[i];
    }
    float avgTAT = (float)totalTAT / n;
    for (i = 0; i < n; i++)
    {
        wt[i] = tat[i] - bt[i];
        totalWT += wt[i];
    }
    float avgWT = (float)totalWT / n;
    printf("\nPNo\tAT\tBT\tCT\tTAT\tWT\n");
    for (i = 0; i < n; i++)
    {
        printf("P%d\t%d\t%d\t%d\t%d\t%d\n", i + 1, at[i], bt[i], ct[i], tat[i],
wt[i]);
    }
    printf("\n");
    printf("Average Waiting time = %f\n", avgWT);
    printf("Average Turn Around time = %f\n", avgTAT);
}
```

Output (FCFS)

```
Enter the number of processes: 6
Enter the arrival times of n processes: 0 1 2 3 4 5
Enter the burst times of n processes: 8 4 2 1 3 2

PNo     AT      BT      CT      TAT     WT
P1      0       8       8       8       0
P2      1       4       12      11      7
P3      2       2       14      12      10
P4      3       1       15      12      11
P5      4       3       18      14      11
P6      5       2       20      15      13

Average Waiting time = 8.666667
Average Turn Around time = 12.000000
```

Program (SJF)

```
#include <stdio.h>
typedef struct
{
    int id,bt,wt,tat,ct,at;
}pcb;
void sortWithBurstTime(pcb pr[],int n)
{
    int i,j;
    pcb temp;
    for(i=0;i<n-1;i++)
    {
        for(j=0;j<n-i-1;j++)
        {
            if(pr[j].bt >pr[j+1].bt)
            {
                temp=pr[j];
                pr[j]= pr[j+1];
                pr[j+1]=temp;
            }
        }
    }
}
int main()
{
    int n,i;
    printf("Enter the number of processes: ");
    scanf("%d",&n);
    pcb pr[n];
    int bt[n],at[n],wt[n],tat[n],ct[n];
    printf("Enter id the burst times of n processes: \n");
    int sum=0;
    for(i=0;i<n;i++)
    {
        scanf("%d",&pr[i].id);
        scanf("%d",&pr[i].bt);
        pr[i].at=0;
    }
    sortWithBurstTime(pr,n);
    for(i=0;i<n;i++)
    {
        sum+=pr[i].bt;
        pr[i].ct=sum;
    }
    int totalTAT=0,totalWT=0;
    for(i=0;i<n;i++)
    {
        pr[i].tat = pr[i].ct-0;
        totalTAT+=pr[i].tat;
    }
    float avgTAT = (float)totalTAT/n;
    for(i=0;i<n;i++)
    {
        pr[i].wt = pr[i].tat-pr[i].bt;
        totalWT+=pr[i].wt;
    }
    float avgWT = (float)totalWT/n;
    printf("\nPNo\tAT\tBT\tCT\tTAT\tWT\n");
    for(i=0;i<n;i++)
    {
        printf("P%d\t%d\t%d\t%d\t%d\t%d\
n",pr[i].id,pr[i].at,pr[i].bt,pr[i].ct,pr[i].tat,pr[i].wt);
    }
```

```
    printf("\n");
    printf("Average Waiting time = %f\n",avgWT);
    printf("Average Turn Around time = %f\n",avgTAT);
}
```

Output (SJF)

```
Enter the number of processes: 5
Enter id the burst times of n processes:
1 4
2 3
3 7
4 1
5 2

PNo         AT      BT      CT      TAT       WT
P4          0       1       1       1         0
P5          0       2       3       3         1
P2          0       3       6       6         3
P1          0       4       10      10        6
P3          0       7       17      17        10

Average Waiting time = 4.000000
Average Turn Around time = 7.400000
```

Program (SRTF)

```
 #include <stdio.h>
typedef struct
{
    int id, at, bt, ct, tat, wt, rt;
} process;
int main()
{
    int i, n, currentTime = 0;
    float avgTAT, avgWT;
    printf("Enter the no of processes: ");
    scanf("%d", &n);
    process a[n];
    printf("Enter id, arrival time and burst time of the processes: \n");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &a[i].id);
        scanf("%d", &a[i].at);
        scanf("%d", &a[i].bt);
        a[i].rt = a[i].bt;
    }
    int completed = 0;
    int totalWT = 0;
    int totalTAT = 0;
    while (completed < n)
    {
        int shortestJob = -1;
        int shortestTime = 99999;
        for (i = 0; i < n; i++)
        {
            if (a[i].at <= currentTime && a[i].rt < shortestTime && a[i].rt > 0)
            {
                shortestJob = i;
                shortestTime = a[i].rt;
            }
        }

        if (shortestJob == -1)
        {
            currentTime++;
        }
        else
        {
            a[shortestJob].rt--;
            currentTime++;
            if (a[shortestJob].rt == 0)
            {
                completed++;
                a[shortestJob].ct = currentTime;
                a[shortestJob].tat = currentTime - a[shortestJob].at;
                a[shortestJob].wt = a[shortestJob].tat - a[shortestJob].bt;
                totalTAT += a[shortestJob].tat;
                totalWT += a[shortestJob].wt;
            }
        }
    }
    avgTAT = (float)totalTAT / n;
    avgWT = (float)totalWT / n;
    printf("\nID\tAT\tBT\tCT\tTAT\tWT");
    for (i = 0; i < n; i++)
    {
        printf("\nP%d\t%d\t%d\t%d\t%d\t%d", a[i].id, a[i].at, a[i].bt, a[i].ct,
a[i].tat, a[i].wt);
```

```
    }
    printf("\nAverage Waiting Time = %f", avgWT);
    printf("\nAverage Turn Around Time = %f\n", avgTAT);
}
```

Output (SRTF)

```
Enter the no of processes: 4
Enter id, arrival time and burst time of the processes:
1 0 3
2 1 6
3 4 4
4 6 2

ID          AT          BT          CT          TAT     WT
P1          0           3           3           3       0
P2          1           6           15          14      8
P3          4           4           8           4       0
P4          6           2           10          4       2

Average Waiting Time = 2.500000
Average Turn Around Time = 6.250000
```

Program (Priority)

```
#include <stdio.h>
typedef struct
{
    int id, at, bt, ct, tat, wt, rt, priority;
} process;
int main()
{
    int n, currentTime = 0;
    float avgTAT, avgWT;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    process a[n];
    printf("Enter the id,  arrival time and burst time of n processes: \n");
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &a[i].id);
        scanf("%d", &a[i].at);
        scanf("%d", &a[i].bt);
        scanf("%d", &a[i].priority);
        a[i].rt = a[i].bt;
    }
    int completed = 0;
    int totalWT = 0;
    int totalTAT = 0;
    while (completed < n)
    {
        int sJob = -1;
        int sPriority = 99999;
        for (int i = 0; i < n; i++)
        {
            if (a[i].at <= currentTime && a[i].priority < sPriority && a[i].rt >
0)
            {
                sJob = i;
                sPriority = a[i].priority;
            }
        }
        if (sJob == -1)
        {
            currentTime++;
        }
        else
        {
            a[sJob].rt--;
            currentTime++;
            if (a[sJob].rt == 0)
            {
                completed++;
                a[sJob].ct = currentTime;
                a[sJob].tat = currentTime - a[sJob].at;
                a[sJob].wt = a[sJob].tat - a[sJob].bt;
                totalTAT += a[sJob].tat;
                totalWT += a[sJob].wt;
            }
        }
    }
    avgTAT = (float)totalTAT / n;
    avgWT = (float)totalWT / n;
    printf("\nID\tAT\tBT\tPrio\tCT\tTAT\tWT");
    for (int i = 0; i < n; i++)
    {
```

```
        printf("\nP%d\t%d\t%d\t%d\t%d\t%d\t%d", a[i].id, a[i].at, a[i].bt,
a[i].priority, a[i].ct, a[i].tat, a[i].wt);
    }
    printf("\nAverage Waiting Time = %f", avgWT);
    printf("\nAverage Turn Around Time = %f\n", avgTAT);
}
```

Output (Priority)

```
Enter the number of processes: 5
Enter the id,  arrival time and burst time of n processes:
1 0 5 6
2 1 2 4
3 2 4 3
4 3 1 1
5 4 7 2
```

ID	AT	BT	Prio	CT	TAT	WT
P1	0	5	6	19	19	14
P2	1	2	4	15	14	12
P3	2	4	3	14	12	8
P4	3	1	1	4	1	0
P5	4	7	2	11	7	0

```
Average Waiting Time = 6.800000
Average Turn Around Time = 10.600000
```

Program (LJF)

```
#include <stdio.h>
typedef struct
{
    int id, bt, wt, tat, ct, at;
} pcb;
void sortWithBurstTime(pcb pr[], int n)
{
    int i, j;
    pcb temp;
    for (i = 0; i < n - 1; i++)
    {
        for (j = 0; j < n - i - 1; j++)
        {
            if (pr[j].bt > pr[j + 1].bt)
            {
                temp = pr[j];
                pr[j] = pr[j + 1];
                pr[j + 1] = temp;
            }
        }
    }
}
int main()
{
    int n, i;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    pcb pr[n];
    int bt[n], at[n], wt[n], tat[n], ct[n];
    printf("Enter id the burst times of n processes: \n");
    int sum = 0;
    for (i = 0; i < n; i++)
    {
        scanf("%d", &pr[i].id);
        scanf("%d", &pr[i].bt);
        pr[i].at = 0;
    }
    sortWithBurstTime(pr, n);
    for (i = 0; i < n; i++)
    {
        sum += pr[i].bt;
        pr[i].ct = sum;
    }
    int totalTAT = 0, totalWT = 0;
    for (i = 0; i < n; i++)
    {
        pr[i].tat = pr[i].ct - 0;
        totalTAT += pr[i].tat;
    }
    float avgTAT = (float)totalTAT / n;
    for (i = 0; i < n; i++)
    {
        pr[i].wt = pr[i].tat - pr[i].bt;
        totalWT += pr[i].wt;
    }
    float avgWT = (float)totalWT / n;
    printf("\nPNo\tAT\tBT\tCT\tTAT\tWT\n");
    for (i = 0; i < n; i++)
    {
        printf("P%d\t%d\t%d\t%d\t%d\t%d\n", pr[i].id, pr[i].at, pr[i].bt,
pr[i].ct, pr[i].tat, pr[i].wt);
    }
```

```
    printf("\n");
    printf("Average Waiting time = %f\n", avgWT);
    printf("Average Turn Around time = %f\n", avgTAT);
}
```

Output (LJF)

```
Enter the number of processes: 5
Enter id the burst times of n processes:
1 4
2 3
3 7
4 1
5 2

PNo     AT        BT          CT          TAT         WT
P3      0         7           7           7           0
P1      0         4           11          11          7
P2      0         3           14          14          11
P5      0         2           16          16          14
P4      0         1           17          17          16

Average Waiting time = 9.600000
Average Turn Around time = 13.000000
```

Program (semaphore)

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

int mutex = 1;
int empty = 5, full = 0;
int count = 0;

int wait(int S)
{
    while (S <= 0)
        ;
    return --S;
}

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

void producer()
{
    mutex = wait(mutex);
    full = signal(full);
    empty = wait(empty);
    printf("produced item %d", ++count);
    mutex = signal(mutex);
}

void consumer()
{
    mutex = wait(mutex);
    full = wait(full);
    empty = signal(empty);
    printf("consumed item %d", count--);
    mutex = signal(mutex);
}

int main()
{
    int n;
    printf("\n1. Producer");
    printf("\n2. Consumer");
    printf("\n3. Exit");
    while (1)
    {
        printf("\nEnter the 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);
        default:
            printf("Enter valid choice");
            break;
        }
        printf("\n\n");
    }
}
```

Output (semaphore)

```
1. Producer
2. Consumer
3. Exit

Enter the choice: 1
produced item 1

Enter the choice: 1
produced item 2

Enter the choice: 1
produced item 3

Enter the choice: 1
produced item 4

Enter the choice: 1
produced item 5

Enter the choice: 1
Buffer is full

Enter the choice: 2
consumed item 5

Enter the choice: 2
consumed item 4

Enter the choice: 2
consumed item 3

Enter the choice: 2
consumed item 2

Enter the choice: 2
consumed item 1

Enter the choice: 2
Buffer is empty
```

Program (First Fit)

```
#include <stdio.h>
void firstfit(int block[], int m, int process[], int n)
{
    int allocation[n];
    for (int i = 0; i < n; i++)
    {
        allocation[i] = -1;
    }
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            if (block[j] >= process[i])
            {
                allocation[i] = j;
                block[j] -= process[i];
                break;
            }
        }
    }
    printf("\nPNo\tPSize\tBlockNo");
    for (int i = 0; i < n; i++)
    {
        printf("\n%d\t%d\t", i + 1, process[i]);
        if (allocation[i] != -1)
        {
            printf("%d", allocation[i] + 1);
        }
        else
        {
            printf("Not Allocated");
        }
    }
    printf("\n");
}
int main()
{
    int m, n;
    printf("Enter the no of blocks: ");
    scanf("%d", &m);
    int block[m];
    printf("Enter size of %d blocks: ");
    for (int i = 0; i < m; i++)
    {
        scanf("%d", block + i);
    }
    printf("Enter the no of processes: ");
    scanf("%d", &n);
    int process[n];
    printf("Enter size of %d processes: ");
    for (int i = 0; i < n; i++)
    {
        scanf("%d", process + i);
    }
    firstfit(block, m, process, n);
}
```

Output (first fit)

```
Enter the no of blocks: 5
Enter size of 5 blocks: 100 500 200 300 600
Enter the no of processes: 4
Enter size of 4 processes: 212 417 112 426

PNo     PSize   BlockNo
1       212     2
2       417     5
3       112     2
4       426     Not Allocated
```

Program (best fit)

```
#include <stdio.h>
void bestfit(int block[], int m, int process[], int n)
{
    int bestBlock;
    int allocation[n];
    for(int i=0;i<n;i++)
    {
        allocation[i]=-1;
    }
    for(int i=0;i<n;i++)
    {
        bestBlock=-1;
        for(int j=0;j<m;j++)
        {
            if(block[j]>=process[i])
            {
                if(bestBlock==-1)
                {
                    bestBlock=j;
                }
                else if(block[j]<block[bestBlock])
                {
                    bestBlock=j;
                }
            }
        }
        if(bestBlock!=-1)
        {
            allocation[i]=bestBlock;
            block[bestBlock]-=process[i];
        }
    }
    printf("\nPNo\tPSize\tBlockNo");
    for(int i=0;i<n;i++)
    {
        printf("\n%d\t%d\t",i+1,process[i]);
        if(allocation[i]!=-1)
        {
            printf("%d",allocation[i]+1);
        }
        else
        {
            printf("Not Allocated");
        }
    }
    printf("\n");
}
int main()
{
    int m,n;
    printf("Enter the no of blocks: ");
    scanf("%d",&m);
    int block[m];
    printf("Enter size of %d blocks: ");
    for(int i=0;i<m;i++)
    {
        scanf("%d",block+i);
    }
    printf("Enter the no of processes: ");
    scanf("%d",&n);
    int process[n];
    printf("Enter size of %d processes: ");
```

```
    for(int i=0;i<n;i++)
    {
        scanf("%d",process+i);
    }
    bestfit(block,m,process,n);
}
```

Output (best fit)

```
Enter the no of blocks: 5
Enter size of 5 blocks: 100 500 200 300 600
Enter the no of processes: 4
Enter size of 4 processes: 212 417 112 426

PNo     PSize   BlockNo
1       212     4
2       417     2
3       112     3
4       426     5
```

Program (worst fit)

```
#include <stdio.h>
void worstfit(int block[], int m, int process[], int n)
{
    int bestBlock;
    int allocation[n];
    for (int i = 0; i < n; i++)
    {
        allocation[i] = -1;
    }
    for (int i = 0; i < n; i++)
    {
        bestBlock = -1;
        for (int j = 0; j < m; j++)
        {
            if (block[j] >= process[i])
            {
                if (bestBlock == -1)
                {
                    bestBlock = j;
                }
                else if (block[j] > block[bestBlock])
                {
                    bestBlock = j;
                }
            }
        }
        if (bestBlock != -1)
        {
            allocation[i] = bestBlock;
            block[bestBlock] -= process[i];
        }
    }
    printf("\nPNo\tPSize\tBlockNo");
    for (int i = 0; i < n; i++)
    {
        printf("\n%d\t%d\t", i + 1, process[i]);
        if (allocation[i] != -1)
        {
            printf("%d", allocation[i] + 1);
        }
        else
        {
            printf("Not Allocated");
        }
    }
    printf("\n");
}
int main()
{
    int m, n;
    printf("Enter the no of blocks: ");
    scanf("%d", &m);
    int block[m];
    printf("Enter size of %d blocks: ");
    for (int i = 0; i < m; i++)
    {
        scanf("%d", block + i);
    }
    printf("Enter the no of processes: ");
    scanf("%d", &n);
    int process[n];
    printf("Enter size of %d processes: ");
```

```
    for (int i = 0; i < n; i++)
    {
        scanf("%d", process + i);
    }
    worstfit(block, m, process, n);
}
```

Output (worst fit)

```
Enter the no of blocks: 5
Enter size of 5 blocks: 100 500 200 300 600
Enter the no of processes: 4
Enter size of 4 processes: 212 417 112 426

PNo         PSize   BlockNo
1           212     5
2           417     2
3           112     5
4           426     Not Allocated
```

Program (round robin)

```
#include <stdio.h>
int q[100], front = -1, rear = -1;

struct process{
    int id, at, tat, bt, wt, rt, ct, status;
} p[20];

void enqueue(int j){
    if (front == -1 && rear == -1)
        front++;
    rear++;
    q[rear] = j;}

int dequeue(){
    if (front == -1)
        return -1;
    int item;
    item = q[front];
    if (front == rear){
        front = -1;
        rear = -1;
    }
    else{
        front++;
    }
    return (item);}

void main()
{
    int n, quantum, time = 0, completed = 0, current = 0, i, dequeuedItem = -1,
qExpire = 0;
    float totalWT = 0, totalTAT = 0;
    printf("Enter the number of processes : ");
    scanf("%d", &n);
    printf("Enter the id,  arrival time and burst time of n processes: \n");
    for (i = 0; i < n; i++) // Input process details
    {
        p[i].id = i + 1;
        p[i].status = 0;
        scanf("%d", &p[i].at);
        scanf("%d", &p[i].bt);
        p[i].rt = p[i].bt;
    }
    printf("\nEnter time quantum : ");
    scanf("%d", &quantum);

    // Waiting for first process to arrive
    while (time != p[0].at)
    {
        time++;
    }
    enqueue(0);
    p[0].status = 1;

    while (completed < n)
    {
        if (dequeuedItem == -1)
        {
            dequeuedItem = dequeue();
        }

        time++;
```

```
        // Loop to check for new process
        for (i = 0; i < n; i++)
        {
            if (p[i].status == 0 && p[i].at <= time)
            {
                enqueue(i);
                p[i].status = 1;
            }
        }

        if (dequeuedItem != -1)
        {
            if (qExpire != quantum && p[dequeuedItem].rt > 0)
            {
                p[dequeuedItem].rt--;
                qExpire++;
            }
            if (p[dequeuedItem].rt == 0)
            {
                p[dequeuedItem].ct = time;
                p[dequeuedItem].tat = p[dequeuedItem].ct - p[dequeuedItem].at;
                p[dequeuedItem].wt = p[dequeuedItem].tat - p[dequeuedItem].bt;
                completed++;
                totalWT += p[dequeuedItem].wt;
                totalTAT += p[dequeuedItem].tat;
                dequeuedItem = -1;
                qExpire = 0;
            }
            else if (qExpire == quantum)
            {
                enqueue(dequeuedItem);
                dequeuedItem = -1;
                qExpire = 0;
            }
        }
    }

    printf("\nID\tAT\tBT\tCT\tTAT\tWT");
    for (i = 0; i < n; i++)
    {
        printf("\nP%d\t%d\t%d\t%d\t%d\t%d", p[i].id, p[i].at, p[i].bt, p[i].ct,
p[i].tat, p[i].wt);
    }
    printf("\n\nAverage Waiting Time : %f", (totalWT / n));
    printf("\nAverage Turn Around Time : %f\n", (totalTAT / n));
}
```

Output (round robin)

```
Enter the number of processes : 5
Enter the id,  arrival time and burst time of n processes:
0 5
1 3
2 1
3 2
4 3

Enter time quantum : 2

ID      AT      BT      CT      TAT     WT
P1      0       5       13      13      8
P2      1       3       12      11      8
P3      2       1       5       3       2
P4      3       2       9       6       4
P5      4       3       14      10      7

Average Waiting Time : 5.800000
Average Turn Around Time : 8.600000
```

Program (IPC)

```
//Reader.c

#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>

void main()
{
   int key = ftok("shm",2);
   int shmid = shmget(key,1024,0666 |
IPC_CREAT);
   char * shmaddr =
shmat(shmid,NULL,0);
   printf("Data read: \n");
   puts(shmaddr);
   exit(0);
}
```

Output

```
Data read:
welcome
```

```
//Writer.c

#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>

void main()
{
    int key = ftok("shm",2);
    int shmid = shmget(key,1024,0666 |
            IPC_CREAT);
    char * shmaddr =
            shmat(shmid,NULL,0);
    printf("Data to store: ");
    fgets(shmaddr,50,stdin);
    shmdt(shmaddr);
    exit(0);
}
```

Output

```
Data to store: welcome
```

Program (Bankers)

```
#include <stdio.h>
int max[100][100];
int alloc[100][100];
int need[100][100];
int avail[100];
int n, r;
void input();
void show();
void cal();
int main()
{
    int i, j;
    printf("Enter the no of Processes\t");
    scanf("%d", &n);
    printf("Enter the no of resources instances\t");
    scanf("%d", &r);
    printf("Enter the Max Matrix\n");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < r; j++)
        {
            scanf("%d", &max[i][j]);
        }
    }
    printf("Enter the Allocation Matrix\n");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < r; j++)
        {
            scanf("%d", &alloc[i][j]);
        }
    }
    printf("Enter the available Resources\n");
    for (j = 0; j < r; j++)
    {
        scanf("%d", &avail[j]);
    }
    show();
    cal();
    return 0;
}
void show()
{
    int i, j;
    printf("Process\t Allocation Max Available\t");
    for (i = 0; i < n; i++)
    {
        printf("\nP%d\t\t ", i);
        for (j = 0; j < r; j++)
        {
            printf("%d ", alloc[i][j]);
        }
        printf("\t");
        for (j = 0; j < r; j++)
        {
            printf("%d ", max[i][j]);
        }
        printf("\t\t");
        if (i == 0)
        {
            for (j = 0; j < r; j++)
                printf("%d ", avail[j]);
```

```
        }
    }
}
void cal()
{
    int finish[100], temp, need[100][100], flag = 1, k, c1 = 0;
    int safe[100];
    int i, j;
    for (i = 0; i < n; i++)
    {
        finish[i] = 0;
    }
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < r; j++)
        {
            need[i][j] = max[i][j] - alloc[i][j];
        }
    }
    printf("\n");
    while (flag)
    {
        flag = 0;
        for (i = 0; i < n; i++)
        {
            int c = 0;
            for (j = 0; j < r; j++)
            {
                if ((finish[i] == 0) && (need[i][j] <= avail[j]))
                {
                    c++;
                    if (c == r)
                    {
                        for (k = 0; k < r; k++)
                        {
                            avail[k] += alloc[i][k];
                            finish[i] = 1;
                            flag = 1;
                        }
                        printf("P%d->", i);
                    }
                }
            }
        }
    }
    for (i = 0; i < n; i++){
        if (finish[i] == 1)
        {
            c1++;
        }
        else
        {
            printf("P%d->", i);
        }
    }
    if (c1 == n){
        printf("\n The system is in safe state");
    }
    else
    {
        printf("\n Process are in dead lock");
        printf("\n System is in unsafe state");
    }
}
```

Output

```
Enter the no of Processes: 5
Enter the no of resources instances: 3
Enter the Max 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

Enter the available Resources
3 3 2
Process  Allocation     Max         Available
P0       0 1 0          7 5 3       3 3 2
P1       2 0 0          3 2 2
P2       3 0 2          9 0 2
P3       2 1 1          2 2 2
P4       0 0 2          4 3 3

P1->P3->P4->P0->P2->
The system is in safe state
```

Program (Disk Scheduling – FCFS)

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 25
int n, head, seekCount, tracks[MAX];
void fcfsdisk()
{
    int curr_track, distance;
    seekCount = 0;
    for (int i = 0; i < n; i++)
    {
        curr_track = tracks[i];
        distance = abs(head - curr_track);
        seekCount += distance;
        head = curr_track;
    }
}
int main()
{
    printf("\n FCFS Disk Scheduling\n");
    printf("\n Enter the number of tracks to be seeked : ");
    scanf("%d", &n);
    if (n > MAX)
    {
        printf("\n Number of tracks to be seeked cannot exceed %d. Exiting...\
n", MAX);
        exit(0);
    }
    printf("\n Enter the starting position of the head : ");
    scanf("%d", &head);
    printf("\n Enter the tracks to be seeked : ");
    for (int i = 0; i < n; i++)
        scanf("%d", &tracks[i]);
    fcfsdisk();
    printf("\n The Seek Sequence is : ");
    for (int i = 0; i < n - 1; i++)
        printf(" %d -> ", tracks[i]);
    printf(" %d\n", tracks[n - 1]);
    printf("\n The Seek Count is : %d\n", seekCount);
    return (0);
}
```

Output

```
FCFS Disk Scheduling

Enter the number of tracks to be seeked : 3

Enter the starting position of the head : 5

Enter the tracks to be seeked : 2 9 4

The Seek Sequence is :  2 ->  9 ->  4

The Seek Count is : 15
```

Program (Disk Scheduling – SCAN)

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 25
int n, head, size, seekCount, tracks[MAX], sequence[MAX];
char dir;
void sort(int arr[], int m)
{
    int temp;
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < m - 1 - i; j++)
        {
            if (arr[j] > arr[j + 1])
            {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
void scandisk()
{
    int currTrack, distance, l = 0, r = 0, left[MAX], right[MAX];
    seekCount = 0;
    if (dir == 'L')
    {
        left[0] = 0;
        l++;
    }
    else if (dir == 'R')
    {
        right[0] = size - 1;
        r++;
    }
    for (int i = 0; i < n; i++)
    {
        if (tracks[i] < head)
            left[l++] = tracks[i];
        if (tracks[i] > head)
            right[r++] = tracks[i];
    }
    sort(left, l);
    sort(right, r);
    int run = 2, x = 0;
    while (run-- > 0)
    {
        if (dir == 'L')
        {
            for (int i = l - 1; i >= 0; i--)
            {
                currTrack = left[i];
                sequence[x++] = currTrack;
                distance = abs(head - currTrack);
                seekCount += distance;
                head = currTrack;
            }
            dir = 'R';
        }
        else
        {
            for (int i = 0; i < r; i++)
```

```
            {
                currTrack = right[i];
                sequence[x++] = currTrack;
                distance = abs(head - currTrack);
                seekCount += distance;
                head = currTrack;
            }
            dir = 'L';
        }
    }
}
int main()
{
    int i;
    printf("\n SCAN Disk Scheduling\n");
    printf("\n Enter the size of the disk : ");
    scanf("%d", &size);
    printf("\n Enter the number of tracks to be seeked : ");
    scanf("%d", &n);
    if (n > MAX)
    {
        printf("\n Number of tracks to be seeked cannot exceed %d Exiting...\n",
MAX);
        exit(0);
    }
    printf("\n Enter the starting position of the head : ");
    scanf("%d", &head);
    if (head > size)
    {
        printf("\n Starting position of head cannot exceed the size of disk.
Exiting...\n");
        exit(0);
    }
    printf("\n Enter the initial direction of the head(L/R) : ");
    scanf(" %c", &dir);
    if ((dir != 'L') && (dir != 'R'))
    {
        printf("\n Invalid direction input. Exiting...\n");
        exit(0);
    }
    printf("\n Enter the tracks to be seeked : ");
    for (int i = 0; i < n; i++)
        scanf("%d", &tracks[i]);
    scandisk();
    printf("\n The Seek Sequence is : ");
    for (i = 0; i < n; i++)
        printf(" %d -> ", sequence[i]);
    printf(" %d\n", sequence[i]);
    printf("\n The Seek Count is : %d\n", seekCount);
    return 0;
}
```

Output

```
SCAN Disk Scheduling
Enter the size of the disk : 10
Enter the number of tracks to be seeked : 3
Enter the starting position of the head : 5
Enter the initial direction of the head(L/R) : R
Enter the tracks to be seeked : 2 6 9
The Seek Sequence is :  6 ->  9 ->  9 ->  2
The Seek Count is : 11
```

Program (disk scheduling c-scan)

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 25
int n, head, size, seekCount, tracks[MAX], sequence[MAX];
void sort(int arr[], int m)
{
    int temp;
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < m - 1 - i; j++)
        {
            if (arr[j] > arr[j + 1])
            {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
void cscandisk()
{
    int currTrack, distance, l, r, left[MAX], right[MAX];
    seekCount = 0;
    l = 0;
    r = 0;
    left[0] = 0;
    l++;
    right[0] = size - 1;
    r++;
    for (int i = 0; i < n; i++)
    {
        if (tracks[i] < head)
            left[l++] = tracks[i];
        if (tracks[i] > head)
            right[r++] = tracks[i];
    }
    sort(left, l);
    sort(right, r);
    int x = 0;
    for (int i = 0; i < r; i++)
    {
        currTrack = right[i];
        sequence[x++] = currTrack;
        distance = abs(head - currTrack);
        seekCount += distance;
        head = currTrack;
    }
    head = 0;
    seekCount += size - 1;
    for (int i = 0; i < l; i++)
    {
        currTrack = left[i];
        sequence[x++] = currTrack;
        distance = abs(head - currTrack);
        seekCount += distance;
        head = currTrack;
    }
}
int main()
{
    int i;
```

```
    printf("\n C-SCAN Disk Scheduling\n");
    printf("\n Enter the size of the disk : ");
    scanf("%d", &size);
    printf("\n Enter the number of tracks to be seeked : ");
    scanf("%d", &n);
    if (n > MAX)
    {
        printf("\n Number of tracks to be seeked cannot exceed %d Exiting...\n",
MAX);
        exit(0);
    }
    printf("\n Enter the starting position of the head : ");
    scanf("%d", &head);
    if (head > size)
    {
        printf("\n Starting position of head cannot exceed the size of disk.
Exiting...\n");
        exit(0);
    }
    printf("\n Enter the tracks to be seeked : ");
    for (int i = 0; i < n; i++)
        scanf("%d", &tracks[i]);
    cscandisk();
    printf("\n The Seek Sequence is : ");
    for (i = 0; i < n; i++)
        printf(" %d -> ", sequence[i]);
    printf(" %d\n", sequence[i]);
    printf("\n The Seek Count is : %d\n", seekCount);
    return 0;
}
```

Output

```
C-SCAN Disk Scheduling

Enter the size of the disk : 10
Enter the number of tracks to be seeked : 3
Enter the starting position of the head : 5
Enter the tracks to be seeked : 5 7 4
The Seek Sequence is :  7 ->  9 ->  0 ->  4
The Seek Count is : 17
```