# COCSC09-OPERATING SYSTEM

## PRACTICAL FILE

AMOGH GARG

2020UCO1688

NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY, NEW DELHI

# EXPERIMENT-1,2 AND 3

Open

fork.c
~/Desktop/Practical Experiments

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
 int main(){
        fork();
        fork();
        printf("Called fork() system call \n");
        return 0;
 }
```

amogh@Amogh: ~/Desktop/Practical Experiments

```
amogh@Amogh:~/Desktop/Practical Experiments$ gcc -o fork fork.c
amogh@Amogh:~/Desktop/Practical Experiments$ ./fork
Called fork() system call
amogh@Amogh:~/Desktop/Practical Experiments$ Called fork() system call
Called fork() system call
Called fork() system call
```

Open

fork.c
~/Desktop/Practical Experiments

```c
// Fork()
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main() {
   pid_t pid, mypid, myppid;
   pid = getpid();
   printf("Before fork: Process id is %d\n", pid);
   pid = fork();

   if (pid < 0) {
      perror("fork() failure\n");
      return 1;
   }

   // Child process
   if (pid == 0) {
      printf("This is child process\n");
      mypid = getpid();
      myppid = getppid();
      printf("Process id is %d and PPID is %d\n", mypid, myppid);
   } else { // Parent process
     sleep(2);
      printf("This is parent process\n");
      mypid = getpid();
      myppid = getppid();
      printf("Process id is %d and PPID is %d\n", mypid, myppid);
      printf("Newly created process id or child pid is %d\n", pid);
   }
   return 0;
}
```

amogh@Amogh: ~/Desktop/Practical Experiments

```
amogh@Amogh:~/Desktop/Practical Experiments$ gcc -o fork fork.c
amogh@Amogh:~/Desktop/Practical Experiments$ ./fork
Before fork: Process id is 2495
This is child process
Process id is 2496 and PPID is 2495
This is parent process
Process id is 2495 and PPID is 2448
Newly created process id or child pid is 2496
amogh@Amogh:~/Desktop/Practical Experiments$
```

Firefox Web Browser

wait.c
~/Desktop/Practical Experiments

```c
// C program to demonstrate working of wait()
#include<stdio.h>
#include<sys/wait.h>
#include<unistd.h>

int main()
{
    if (fork()== 0)
        printf("HC: hello from child\n");
    else
    {
        printf("HP: hello from parent\n");
        wait(NULL);
        printf("CT: child has terminated\n");
    }

    printf("Bye\n");
    return 0;
}
```

amogh@Amogh: ~/Desktop/Practical Experiments

```
amogh@Amogh:~/Desktop/Practical Experiments$ vi wait.c
amogh@Amogh:~/Desktop/Practical Experiments$ gcc -o wait wait.c
amogh@Amogh:~/Desktop/Practical Experiments$ ./wait
HP: hello from parent
HC: hello from child
Bye
CT: child has terminated
Bye
amogh@Amogh:~/Desktop/Practical Experiments$
```

Open

exit.c
~/Desktop/Practical Experiments

```c
// C program to demonstrate working of exit()
#include <stdio.h>
#include <stdlib.h>

void exitfunc() {
   printf("Called cleanup function - exitfunc()\n");
   return;
}

int main() {
   atexit(exitfunc);
   printf("Hello, World!\n");
   exit (0);
}
```

amogh@Amogh: ~/Desktop/Practical Experiments

```
amogh@Amogh:~/Desktop/Practical Experiments$ vi exit.c
amogh@Amogh:~/Desktop/Practical Experiments$ gcc -o exit exit.c
amogh@Amogh:~/Desktop/Practical Experiments$ .\exit
.exit: command not found
amogh@Amogh:~/Desktop/Practical Experiments$ ./exit
Hello, World!
Called cleanup function - exitfunc()
amogh@Amogh:~/Desktop/Practical Experiments$
```

Open

exec.c
~/Desktop/Practical Experiments

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main(){
        char *args[]={"./test",NULL};
        execvp(args[0],args);
        printf("Terminating...");
        return 0;
}
```

amogh@Amogh: ~/Desktop/Practical Experiments

```
amogh@Amogh:~/Desktop/Practical Experiments$ vi exec.c
amogh@Amogh:~/Desktop/Practical Experiments$ gcc -o exec exec.c
amogh@Amogh:~/Desktop/Practical Experiments$ ./exec
Hello World
amogh@Amogh:~/Desktop/Practical Experiments$
```

# EXPERIMENT-4 (FCFS)

```c
//FCFS
#include<stdio.h>
// Function to find the waiting time for all
// processes
void findWaitingTime(int processes[], int n,int bt[], int wt[])
{
        // waiting time for first process is 0
        wt[0] = 0;

        // calculating waiting time
        for (int i = 1; i < n ; i++ )
                wt[i] = bt[i-1] + wt[i-1] ;
}

// Function to calculate turn around time
void findTurnAroundTime( int processes[], int n,int bt[], int wt[], int tat[])
{
        // calculating turnaround time by adding
        // bt[i] + wt[i]
        for (int i = 0; i < n ; i++)
                tat[i] = bt[i] + wt[i];
}

//Function to calculate average time
void findavgTime( int processes[], int n, int bt[])
{
        int wt[n], tat[n], total_wt = 0, total_tat = 0;

        //Function to find waiting time of all processes
        findWaitingTime(processes, n, bt, wt);

        //Function to find turn around time for all processes
        findTurnAroundTime(processes, n, bt, wt, tat);

        //Display processes along with all details
        printf("Processes Burst time Waiting time Turn around time\n");

        // Calculate total waiting time and total turn
        // around time
        for (int i=0; i<n; i++)
        {
                total_wt = total_wt + wt[i];
                total_tat = total_tat + tat[i];
                printf(" %d ",(i+1));
                printf("         %d ", bt[i] );
```

```c
                printf("         %d ", bt[i] );
                printf("         %d",wt[i] );
                printf("         %d\n",tat[i] );
        }
        int s=(float)total_wt / (float)n;
        int t=(float)total_tat / (float)n;
        printf("Average waiting time = %d",s);
        printf("\n");
        printf("Average turn around time = %d ",t);
}

// Driver code
int main()
{
        //process id's
        int processes[] = { 1, 2, 3};
        int n = sizeof processes / sizeof processes[0];

        //Burst time of all processes
        int burst_time[] = {10, 5, 8};

        findavgTime(processes, n, burst_time);
        return 0;
}
```

amogh@Amogh: ~/Desktop/Practical Experiments

```
amogh@Amogh:~/Desktop/Practical Experiments$ gcc -o fcfs fcfs.c
amogh@Amogh:~/Desktop/Practical Experiments$ ./fcfs
Processes Burst time Waiting time Turn around time
 1       10      0       10
 2       5       10      15
 3       8       15      23
Average waiting time = 8
amogh@Amogh:~/Desktop/Practical Experiments$
```

# EXPERIMENT-5 (SJF)

```c
#include<stdio.h>
 int main()
{
    int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
    float avg_wt,avg_tat;
    printf("Enter number of process:");
    scanf("%d",&n);

    printf("\nEnter Burst Time:\n");
    for(i=0;i<n;i++)
    {
        printf("p%d:",i+1);
        scanf("%d",&bt[i]);
        p[i]=i+1;
    }

   //sorting of burst times
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(bt[j]<bt[pos])
                pos=j;
        }

        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;

        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }

    wt[0]=0;


    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];

        total+=wt[i];
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(bt[j]<bt[pos])
                pos=j;
        }

        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;

        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }

    wt[0]=0;


    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];

        total+=wt[i];
    }

    avg_wt=(float)total/n;
    total=0;

    printf("\nProcesst    Burst Time    \tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];
        total+=tat[i];
        printf("\np%d\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }

    avg_tat=(float)total/n;
    printf("\n\nAverage Waiting Time=%f",avg_wt);
    printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

```
amogh@Amogh:~/Desktop/Practical Experiments$ vi sjf.c
amogh@Amogh:~/Desktop/Practical Experiments$ gcc sjf.c -o sjf
amogh@Amogh:~/Desktop/Practical Experiments$ .\sjf
.sjf: command not found
amogh@Amogh:~/Desktop/Practical Experiments$ ./sjf
Enter number of process:3
nEnter Burst Time:np1:2
p2:5
p3:1
nProcesst    Burst Time    tWaiting TimetTurnaround Timenp3tt  1tt    0ttt1np1tt  2tt    1ttt3np2tt  5tt    3ttt8nnAverage Waiting Time=1.333333nAverage Turnaround Time=4.000000
tical Experiments$ gcc sjf.c -o sjf
amogh@Amogh:~/Desktop/Practical Experiments$ ./sjf
Enter number of process:3

Enter Burst Time:
p1:3
p2:2
p3:1

Processt    Burst Time          Waiting Time    Turnaround Time
p3              1                   0                   1
p2              2                   1                   3
p1              3                   3                   6

Average Waiting Time=1.333333
Average Turnaround Time=3.333333
amogh@Amogh:~/Desktop/Practical Experiments$
```

# EXPERIMENT-6 (PREEMPTIVE PRIORITY)

```c
#include<stdio.h>
 int main()
{
    int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
    float avg_wt,avg_tat;
    printf("Enter number of process:");
    scanf("%d",&n);

    printf("\nEnter Burst Time:\n");
    for(i=0;i<n;i++)
    {
        printf("p%d:",i+1);
        scanf("%d",&bt[i]);
        p[i]=i+1;
    }

   //sorting of burst times
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(bt[j]<bt[pos])
                pos=j;
        }

        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;

        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }

    wt[0]=0;


    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];

        total+=wt[i];
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(bt[j]<bt[pos])
                pos=j;
        }

        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;

        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }

    wt[0]=0;


    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];

        total+=wt[i];
    }

    avg_wt=(float)total/n;
    total=0;

    printf("\nProcesst    Burst Time    \tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];
        total+=tat[i];
        printf("\np%d\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }

    avg_tat=(float)total/n;
    printf("\n\nAverage Waiting Time=%f",avg_wt);
    printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

```
amogh@Amogh:~/Desktop/Practical Experiments$ vi priority.c
amogh@Amogh:~/Desktop/Practical Experiments$ gcc priority.c -o priority
amogh@Amogh:~/Desktop/Practical Experiments$ ./priority
Enter the number of the process
3
Enter the arrival time , burst time and priority of the process
AT BT PT
0  4  1
1  2  3
2  3  2
ID WT TAT
1 0     4
2 6     8
3 2     5
Avg waiting time of the process  is 2.666667
Avg turn around time of the process is 5.666667
amogh@Amogh:~/Desktop/Practical Experiments$
```

# EXPERIMENT-7 (ROUND-ROBIN)

```c
#include<stdio.h>

int main()
{
	int i, limit, total = 0, x, counter = 0, time_quantum;
	int wait_time = 0, turnaround_time = 0, arrival_time[10], burst_time[10], temp[10];
	float average_wait_time, average_turnaround_time;
	printf("\nEnter Total Number of Processes:\t");
	scanf("%d", &limit);
	x = limit;
	for(i = 0; i < limit; i++)
	{
		printf("\nEnter Details of Process[%d]\n", i + 1);

		printf("Arrival Time:t");

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

		printf("Burst Time:t");

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

		temp[i] = burst_time[i];
	}

	printf("\nEnter Time Quantum:\t");
	scanf("%d", &time_quantum);
	printf("\nProcess IDttBurst Timet Turnaround Timet Waiting Time\n");
	for(total = 0, i = 0; x != 0;)
	{
		if(temp[i] <= time_quantum && temp[i] > 0)
		{
			total = total + temp[i];
			temp[i] = 0;
			counter = 1;
		}
		else if(temp[i] > 0)
		{
			temp[i] = temp[i] - time_quantum;
			total = total + time_quantum;
		}
		if(temp[i] == 0 && counter == 1)
		{
			x--;
			printf("\nProcess[%d]\t\t%d\t\t %d\t\t\t %d", i + 1, burst_time[i], total - arrival_time[i], total - arrival_time[i] - burst_time[i]);
```


```c
	printf("\nEnter Time Quantum:\t");
	scanf("%d", &time_quantum);
	printf("\nProcess IDttBurst Timet Turnaround Timet Waiting Time\n");
	for(total = 0, i = 0; x != 0;)
	{
		if(temp[i] <= time_quantum && temp[i] > 0)
		{
			total = total + temp[i];
			temp[i] = 0;
			counter = 1;
		}
		else if(temp[i] > 0)
		{
			temp[i] = temp[i] - time_quantum;
			total = total + time_quantum;
		}
		if(temp[i] == 0 && counter == 1)
		{
			x--;
```


```c
			printf("\nProcess[%d]\t\t%d\t\t %d\t\t\t %d", i + 1, burst_time[i], total - arrival_time[i], total - arrival_time[i] - burst_time[i]);
```


```c
```


```c
			wait_time = wait_time + total - arrival_time[i] - burst_time[i];
			turnaround_time = turnaround_time + total - arrival_time[i];
			counter = 0;
		}
		if(i == limit - 1)
		{
			i = 0;
		}
		else if(arrival_time[i + 1] <= total)
		{
			i++;
		}
		else
		{
			i = 0;
		}
	}

	average_wait_time = wait_time * 1.0 / limit;
	average_turnaround_time = turnaround_time * 1.0 / limit;
	printf("\n\nAverage Waiting Time:\t%f", average_wait_time);
	printf("\nAvg Turnaround Time:\t%f\n", average_turnaround_time);
	return 0;
```

```
amogh@Amogh:~/Desktop/Practical Experiments$ vi rr.c
amogh@Amogh:~/Desktop/Practical Experiments$ gcc -o rr rr.c
amogh@Amogh:~/Desktop/Practical Experiments$ ./rr

Enter Total Number of Processes:        3

Enter Details of Process[1]
Arrival Time:t1
Burst Time:t2

Enter Details of Process[2]
Arrival Time:t2
Burst Time:t6

Enter Details of Process[3]
Arrival Time:t4
Burst Time:t7

Enter Time Quantum:     2

Process IDttBurst Timet Turnaround Timet Waiting Time

Process[1]              2               1                       -1
Process[2]              6               10                      4
Process[3]              7               11                      4

Average Waiting Time:   2.333333
Avg Turnaround Time:    7.333333
amogh@Amogh:~/Desktop/Practical Experiments$
```

# EXPERIMENT-8 (MFQS)

Open

mfqs.c
~/Desktop/Practical Experiments

```c
#include<stdio.h>

struct process
{
    char name;
    int AT,BT,WT,TAT,RT,CT;
}Q1[10],Q2[10],Q3[10];/*Three queues*/

int n;
void sortByArrival()
{
struct process temp;
int i,j;
for(i=0;i<n;i++)
    {
        for(j=i+1;j<n;j++)
            {
                if(Q1[i].AT>Q1[j].AT)
                    {
                        temp=Q1[i];
                        Q1[i]=Q1[j];
                        Q1[j]=temp;
                    }
            }
    }
}

int main()
{
     int i,j,k=0,r=0,time=0,tq1=5,tq2=8,flag=0;
     char c;
     printf("Enter no of processes:");
     scanf("%d",&n);
     for(i=0,c='A';i<n;i++,c++)
     {
         Q1[i].name=c;
         printf("\nEnter the arrival time and burst time of process %c: ",Q1[i].name);
         scanf("%d%d",&Q1[i].AT,&Q1[i].BT);
         Q1[i].RT=Q1[i].BT;/*save burst time in remaining time for each process*/

    }
sortByArrival();
time=Q1[0].AT;
printf("Process in first queue following RR with qt=5");
printf("\nProcess\t\tRT\t\tWT\t\tTAT\t\t");
```

amogh@Amogh: ~/Desktop/Practical Experiments

```
amogh@Amogh:~/Desktop/Practical Experiments$ vi mfqs.c
amogh@Amogh:~/Desktop/Practical Experiments$ gcc -o mfqs mfqs.c
amogh@Amogh:~/Desktop/Practical Experiments$ ./mfqs
Enter no of processes:3

Enter the arrival time and burst time of process A: 1 3

Enter the arrival time and burst time of process B: 2 4

Enter the arrival time and burst time of process C: 2 1
Process in first queue following RR with qt=5
Process         RT              WT              TAT
A               3               0               3
B               4               2               6
C               1               6               7amogh@Amogh:~/Desktop/Practica
amogh@Amogh:~/Desktop/Practical Experiments$
```

```c
printf("\nProcess\t\tRT\t\tWT\t\tTAT\t\t");
for(i=0;i<n;i++)
{

  if(Q1[i].RT<=tq1)
  {

        time+=Q1[i].RT;/*from arrival time of first process to completion of this process*/
        Q1[i].RT=0;
        Q1[i].WT=time-Q1[i].AT-Q1[i].BT;/*amount of time process has been waiting in the first queue*/
        Q1[i].TAT=time-Q1[i].AT;/*amount of time to execute the process*/
        printf("\n%c\t\t%d\t\t%d\t\t%d",Q1[i].name,Q1[i].BT,Q1[i].WT,Q1[i].TAT);

  }
  else/*process moves to queue 2 with qt=8*/
  {
      Q2[k].WT=time;
      time+=tq1;
      Q1[i].RT-=tq1;
      Q2[k].BT=Q1[i].RT;
      Q2[k].RT=Q2[k].BT;
      Q2[k].name=Q1[i].name;
      k=k+1;
      flag=1;
   }
}
if(flag==1)
 {printf("\nProcess in second queue following RR with qt=8");
  printf("\nProcess\t\tRT\t\tWT\t\tTAT\t\t");
}for(i=0;i<k;i++)
   {
    if(Q2[i].RT<=tq2)
     {
       time+=Q2[i].RT;/*from arrival time of first process +BT of this process*/
       Q2[i].RT=0;
       Q2[i].WT=time-tq1-Q2[i].BT;/*amount of time process has been waiting in the ready queue*/
       Q2[i].TAT=time-Q2[i].AT;/*amount of time to execute the process*/
       printf("\n%c\t\t%d\t\t%d\t\t%d",Q2[i].name,Q2[i].BT,Q2[i].WT,Q2[i].TAT);

    }
    else/*process moves to queue 3 with FCFS*/
    {
      Q3[r].AT=time;
      time+=tq2;
      Q2[i].RT-=tq2;
```

```
        time+=Q2[i].RT;/*from arrival time of first process +BT of this process*/
        Q2[i].RT=0;
        Q2[i].WT=time-tq1-Q2[i].BT;/*amount of time process has been waiting in the ready queue*/
        Q2[i].TAT=time-Q2[i].AT;/*amount of time to execute the process*/
        printf("\n%c\t\t%d\t\t%d\t\t%d",Q2[i].name,Q2[i].BT,Q2[i].WT,Q2[i].TAT);

    }
    else/*process moves to queue 3 with FCFS*/
    {
      Q3[r].AT=time;
      time+=tq2;
      Q2[i].RT-=tq2;
      Q3[r].BT=Q2[i].RT;
      Q3[r].RT=Q3[r].BT;
      Q3[r].name=Q2[i].name;
      r=r+1;
      flag=2;
    }
  }

{if(flag==2)
printf("\nProcess in third queue following FCFS ");
}
for(i=0;i<r;i++)
{
    if(i==0)
            Q3[i].CT=Q3[i].BT+time-tq1-tq2;
        else
            Q3[i].CT=Q3[i-1].CT+Q3[i].BT;

}

for(i=0;i<r;i++)
    {
        Q3[i].TAT=Q3[i].CT;
        Q3[i].WT=Q3[i].TAT-Q3[i].BT;
        printf("\n%c\t\t%d\t\t%d\t\t%d\t\t",Q3[i].name,Q3[i].BT,Q3[i].WT,Q3[i].TAT);

    }

}
```

# EXPERIMENT-9 (DEADLOCK AVOIDANCE)

Open

deadlock_avoidance.c
~/Desktop/Practical Experiments

```c
// Banker's Algorithm
#include <stdio.h>
int main(){
        // P0, P1, P2, P3, P4 are the Process names here
        int n, m, i, j, k;
        n = 5; // Number of processes
        m = 3; // Number of resources
        int alloc[5][3] = { { 0, 1, 0 }, // P0 // Allocation Matrix
                                        { 2, 0, 0 }, // P1
                                        { 3, 0, 2 }, // P2
                                        { 2, 1, 1 }, // P3
                                        { 0, 0, 2 } }; // P4

        int max[5][3] = { { 7, 5, 3 }, // P0 // MAX Matrix
                                { 3, 2, 2 }, // P1
                                { 9, 0, 2 }, // P2
                                { 2, 2, 2 }, // P3
                                { 4, 3, 3 } }; // P4

        int avail[3] = { 3, 3, 2 }; // Available Resources

        int f[n], ans[n], ind = 0;
        for (k = 0; k < n; k++) {
                f[k] = 0;
        }
        int need[n][m];
        for (i = 0; i < n; i++) {
                for (j = 0; j < m; j++)
                        need[i][j] = max[i][j] - alloc[i][j];
        }
        int y = 0;
        for (k = 0; k < 5; k++) {
                for (i = 0; i < n; i++) {
                        if (f[i] == 0) {

                                int flag = 0;
                                for (j = 0; j < m; j++) {
                                        if (need[i][j] > avail[j]){
                                                flag = 1;
                                                break;
                                        }
                                }

                                if (flag == 0) {
                                        ans[ind++] = i;
```

amogh@Amogh: ~/Desktop/Practical Experiments

```
amogh@Amogh:~/Desktop/Practical Experiments$ vi deadlock_avoidance.c
amogh@Amogh:~/Desktop/Practical Experiments$ gcc -o deadlock_avoidance deadlock
_avoidance.c
amogh@Amogh:~/Desktop/Practical Experiments$ ./deadlock_avoidance
Following is the SAFE Sequence
 P1 -> P3 -> P4 -> P0 -> P2
amogh@Amogh:~/Desktop/Practical Experiments$
```

```c
        int y = 0;
        for (k = 0; k < 5; k++) {
                for (i = 0; i < n; i++) {
                        if (f[i] == 0) {

                                int flag = 0;
                                for (j = 0; j < m; j++) {
                                        if (need[i][j] > avail[j]){
                                                flag = 1;
                                                break;
                                        }
                                }

                                if (flag == 0) {
                                        ans[ind++] = i;
                                        for (y = 0; y < m; y++)
                                                avail[y] += alloc[i][y];
                                        f[i] = 1;
                                }
                        }
                }
        }

        int flag = 1;
        for(int i=0;i<n;i++)
        {
        if(f[i]==0)
        {
                flag=0;
                printf("The following system is not safe\n");
                break;
        }
        }
        if(flag==1)
        {
        printf("Following is the SAFE Sequence\n");
        for (i = 0; i < n - 1; i++)
                printf(" P%d ->", ans[i]);
        printf(" P%d", ans[n - 1]);
        }
        printf("\n");
        return (0);

}
```

# EXPERIMENT-10 (DEADLOCK DETECTION)

```c
#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++)
```


```c
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++)
```


```c
if(mark[i]!=1)
deadlock=1;


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

```
amogh@Amogh:~/Desktop/Practical Experiments$ ./deadlock_detection

Enter the no of process: 4

Enter the no of resources: 5

Total Amount of the Resource R1: 2

Total Amount of the Resource R2: 1

Total Amount of the Resource R3: 1

Total Amount of the Resource R4: 2

Total Amount of the Resource R5: 1

Enter the request matrix:0 1 0 0 1
0 0 1 0 1
0 0 0 0 1
1 0 1 0 1

Enter the allocation matrix:1 0 1 1 0
1 1 0 0 0
0 0 0 1 0
0 0 0 0 0

 Deadlock detectedamogh@Amogh:~/Desktop/Practical Experiments$
```

# EXPERIMENT-11 (BEST-FIT)

```c
#include<stdio.h>

void main()
{
int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999;
static int barray[20],parray[20];
printf("\n\t\t\tMemory Management Scheme - Best Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of processes:");
scanf("%d",&np);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
    {
printf("Block no.%d:",i);
        scanf("%d",&b[i]);
    }
printf("\nEnter the size of the processes :-\n");
for(i=1;i<=np;i++)
    {
        printf("Process no.%d:",i);
        scanf("%d",&p[i]);
    }
for(i=1;i<=np;i++)
{
for(j=1;j<=nb;j++)
{
if(barray[j]!=1)
{
temp=b[j]-p[i];
if(temp>=0)
if(lowest>temp)
{
parray[i]=j;
lowest=temp;
}
}
}
fragment[i]=lowest;
barray[parray[i]]=1;
lowest=10000;
}
printf("\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragment");
for(i=1;i<=np && parray[i]!=0;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,p[i],parray[i],b[parray[i]],fragment[i]);
```


```c
if(lowest>temp)
{
parray[i]=j;
lowest=temp;
}
}
}
fragment[i]=lowest;
barray[parray[i]]=1;
lowest=10000;
}
printf("\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragment");
for(i=1;i<=np && parray[i]!=0;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,p[i],parray[i],b[parray[i]],fragment[i]);
```


```
amogh@Amogh:~/Desktop/Practical Experiments$ vi best_fit.c
amogh@Amogh:~/Desktop/Practical Experiments$ gcc -o best_fit best_fit.c
Thunderbird Mail esktop/Practical Experiments$ ./best_fit

                        Memory Management Scheme - Best Fit
Enter the number of blocks:5
Enter the number of processes:4

Enter the size of the blocks:-
Block no.1:10
Block no.2:15
Block no.3:5
Block no.4:9
Block no.5:3

Enter the size of the processes :-
Process no.1:1
Process no.2:4
Process no.3:7
Process no.4:12

Process_no      Process_size    Block_no        Block_size      Fragment
1               1               5               3               2
2               4               3               5               1
3               7               4               9               2
4               12              2               15              3amogh@Amogh:~/Desktop/Practical Experiments$
```

# EXPERIMENT-12 (FIFO PAGE-REPLACEMENT)

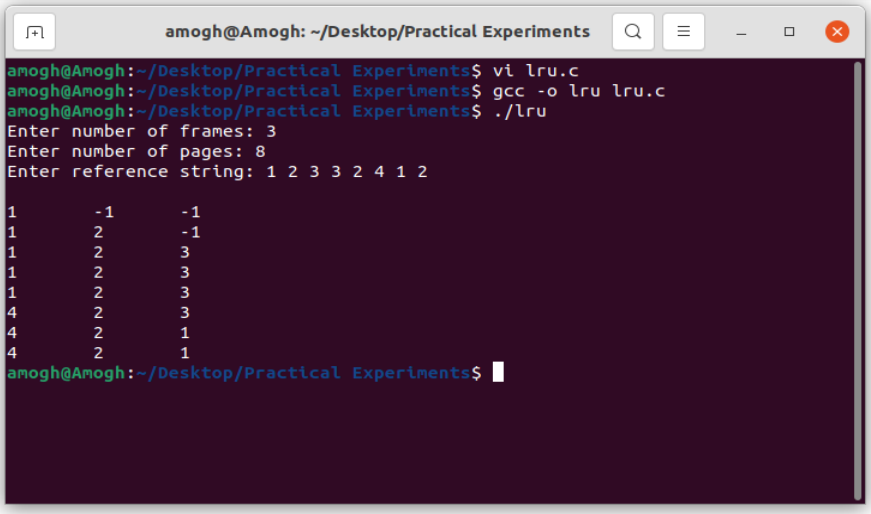
```c
#include <stdio.h>
int main()
{
int referenceString[10], pageFaults = 0, m, n, s, pages, frames;
printf("\nEnter the number of Pages:\t");
scanf("%d", &pages);
printf("\nEnter reference string values:\n");
for( m = 0; m < pages; m++)
{
   printf("Value No. [%d]:\t", m + 1);
   scanf("%d", &referenceString[m]);
}
printf("\n What are the total number of frames:\t");
{
   scanf("%d", &frames);
}
int temp[frames];
for(m = 0; m < frames; m++)
{
  temp[m] = -1;
}
for(m = 0; m < pages; m++)
{
  s = 0;
  for(n = 0; n < frames; n++)
   {
      if(referenceString[m] == temp[n])
         {
            s++;
            pageFaults--;
         }
   }
   pageFaults++;
   if((pageFaults <= frames) && (s == 0))
      {
        temp[m] = referenceString[m];
      }
   else if(s == 0)
      {
        temp[(pageFaults - 1) % frames] = referenceString[m];
      }
      printf("\n");
      for(n = 0; n < frames; n++)
       {
         printf("%d\t", temp[n]);
```

```
amogh@Amogh: ~/Desktop/Practical Experiments
amogh@Amogh:~/Desktop/Practical Experiments$ vi fifo_pr.c
amogh@Amogh:~/Desktop/Practical Experiments$ gcc -o fifo_pr fifo_pr.c
amogh@Amogh:~/Desktop/Practical Experiments$ ./fifo_pr

Enter the number of Pages:      5

Enter reference string values:
Value No. [1]:  5
Value No. [2]:  4
Value No. [3]:  3
Value No. [4]:  2
Value No. [5]:  1

 What are the total number of frames:   4

5       -1      -1      -1
5       4       -1      -1
5       4       3       -1
5       4       3       2
1       4       3       2
Total Page Faults:      5
amogh@Amogh:~/Desktop/Practical Experiments$
```

```c
  s = 0;
  for(n = 0; n < frames; n++)
   {
      if(referenceString[m] == temp[n])
         {
            s++;
            pageFaults--;
         }
   }
   pageFaults++;
   if((pageFaults <= frames) && (s == 0))
      {
        temp[m] = referenceString[m];
      }
   else if(s == 0)
      {
        temp[(pageFaults - 1) % frames] = referenceString[m];
      }
      printf("\n");
      for(n = 0; n < frames; n++)
       {
         printf("%d\t", temp[n]);
       }
}
printf("\nTotal Page Faults:\t%d\n", pageFaults);
return 0;
}
```

# EXPERIMENT-13 (LRU PAGE REPLACEMENT)

```c
//LRU
#include <stdio.h>
int findLRU(int time[], int n)
{
   int i, minimum = time[0], pos = 0;

   for (i = 1; i < n; ++i)
   {
      if (time[i] < minimum)
      {
         minimum = time[i];
         pos = i;
      }
   }

   return pos;
}

//main function
int main()
{
   int no_of_frames, no_of_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j, pos, faults = 0;
   printf("Enter number of frames: ");
   scanf("%d", &no_of_frames);

   printf("Enter number of pages: ");
   scanf("%d", &no_of_pages);

   printf("Enter reference string: ");

   for (i = 0; i < no_of_pages; ++i)
   {
      scanf("%d", &pages[i]);
   }

   for (i = 0; i < no_of_frames; ++i)
   {
      frames[i] = -1;
   }

   for (i = 0; i < no_of_pages; ++i)
   {
      flag1 = flag2 = 0;

      for (j = 0; j < no_of_frames; ++j)
      {
         if (frames[j] == pages[i])
         {
            counter++;
            time[j] = counter;
            flag1 = flag2 = 1;
            break;
         }
      }

      if (flag1 == 0)
      {
         for (j = 0; j < no_of_frames; ++j)
         {
            if (frames[j] == -1)
            {
               counter++;
               faults++;
               frames[j] = pages[i];
               time[j] = counter;
               flag2 = 1;
               break;
            }
         }
      }
      if (flag2 == 0)
      {
         pos = findLRU(time, no_of_frames);
         counter++;
         faults++;
         frames[pos] = pages[i];
         time[pos] = counter;
      }

      printf("\n");

      for (j = 0; j < no_of_frames; ++j)
      {
         printf("%d\t", frames[j]);
      }
   }
   printf("\nTotal Page Faults = %d", faults);
   return 0;
}
```



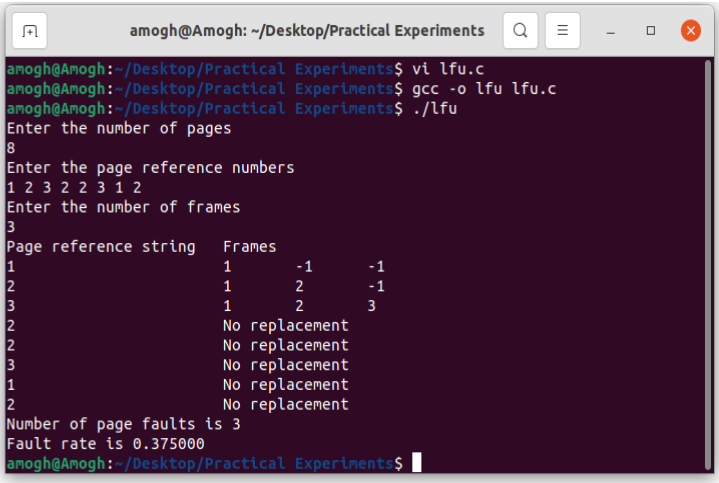
# EXPERIMENT-14 AND 15 (SECOND CHANCE)

```c
#include<stdio.h>
int n,nf;
int in[100];
int p[50];
int hit=0;
int i,j,k;
int pgfaultcnt=0;

void getData()
{
    printf("\nEnter length of page reference sequence:");
    scanf("%d",&n);
    printf("\nEnter the page reference sequence:");
    for(i=0; i<n; i++)
        scanf("%d",&in[i]);
    printf("\nEnter no of frames:");
    scanf("%d",&nf);
}

void initialize()
{
    pgfaultcnt=0;
    for(i=0; i<nf; i++)
        p[i]=9999;
}

int isHit(int data)
{
    hit=0;
    for(j=0; j<nf; j++)
    {
        if(p[j]==data)
        {
            hit=1;
            break;
        }

    }

    return hit;
}

int getHitIndex(int data)
{
    int hitind;
    for(k=0; k<nf; k++)
    {
        if(p[k]==data)
        {
            hitind=k;
            break;
        }
    }
    return hitind;
}

void dispPages()
{
    for (k=0; k<nf; k++)
    {
        if(p[k]!=9999)
            printf(" %d",p[k]);
    }

}

void dispPgFaultCnt()
{
    printf("\nTotal no of page faults:%d",pgfaultcnt);
}
void secondchance()
{
    int usedbit[50];
    int victimptr=0;
    initialize();
    for(i=0; i<nf; i++)
        usedbit[i]=0;
    for(i=0; i<n; i++)
    {
        printf("\nFor %d:",in[i]);
        if(isHit(in[i]))
        {
            printf("No page fault!");
            int hitindex=getHitIndex(in[i]);
            if(usedbit[hitindex]==0)
                usedbit[hitindex]=1;
        }
        else
        {
            pgfaultcnt++;
            if(usedbit[victimptr]==1)
            {
                do
                {
                    usedbit[victimptr]=0;
                    victimptr++;
                    if(victimptr==nf)
                        victimptr=0;
                }
                while(usedbit[victimptr]!=0);
            }
            if(usedbit[victimptr]==0)
            {
                p[victimptr]=in[i];
                usedbit[victimptr]=1;
                victimptr++;
            }
            dispPages();

        }
        if(victimptr==nf)
            victimptr=0;
    }
    dispPgFaultCnt();
}
int main(){
        secondchance();
        return 0;
}
```

# EXPERIMENT-16 (LFU PAGE REPLACEMENT)

```c
#include<stdio.h>
void print(int frameno,int frame[])
{
            int j;
            for(j=0;j<frameno;j++)
            printf("%d\t",frame[j]);
            printf("\n");
}
int main()
{
            int i,j,k,n,page[50],frameno,frame[10],move=0,flag,count=0,count1[10]={0},
                        repindex,leastcount;
            float rate;
            printf("Enter the number of pages\n");
            scanf("%d",&n);
            printf("Enter the page reference numbers\n");
            for(i=0;i<n;i++)
            scanf("%d",&page[i]);
            printf("Enter the number of frames\n");
            scanf("%d",&frameno);
            for(i=0;i<frameno;i++)
            frame[i]=-1;
            printf("Page reference string\tFrames\n");
            for(i=0;i<n;i++)
            {
                        printf("%d\t\t\t",page[i]);
                        flag=0;
                        for(j=0;j<frameno;j++)
                        {
                                    if(page[i]==frame[j])
                                    {
                                                flag=1;
                                                count1[j]++;
                                                printf("No replacement\n");
                                                break;
                                    }
                        }
                        if(flag==0&&count<frameno)
                        {
                                    frame[move]=page[i];
                                    count1[move]=1;
                                    move=(move+1)%frameno;
                                    count++;
                                    print(frameno,frame);
                        }
```

```c
            {
                        printf("%d\t\t\t",page[i]);
                        flag=0;
                        for(j=0;j<frameno;j++)
                        {
                                    if(page[i]==frame[j])
                                    {
                                                flag=1;
                                                count1[j]++;
                                                printf("No replacement\n");
                                                break;
                                    }
                        }
                        if(flag==0&&count<frameno)
                        {
                                    frame[move]=page[i];
                                    count1[move]=1;
                                    move=(move+1)%frameno;
                                    count++;
                                    print(frameno,frame);
                        }
                        else if(flag==0)
                        {
                                    repindex=0;
                                    leastcount=count1[0];
                                    for(j=1;j<frameno;j++)
                                    {
                                                if(count1[j]<leastcount)
                                                {
                                                            repindex=j;
                                                            leastcount=count1[j];
                                                }
                                    }

                                    frame[repindex]=page[i];
                                    count1[repindex]=1;
                                    count++;
                                    print(frameno,frame);
                        }
            }
            rate=(float)count/(float)n;
            printf("Number of page faults is %d\n",count);
            printf("Fault rate is %f\n",rate);
            return 0;
}
```



# EXPERIMENT-17 (FCFS DISK SCHEDULING)

```c
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int RQ[100],i,n,TotalHeadMoment=0,initial;
    printf("Enter the number of Requests\n");
    scanf("%d",&n);
    printf("Enter the Requests sequence\n");
    for(i=0;i<n;i++)
     scanf("%d",&RQ[i]);
    printf("Enter initial head position\n");
    scanf("%d",&initial);

    // logic for FCFS disk scheduling

    for(i=0;i<n;i++)
    {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
        initial=RQ[i];
    }

    printf("Total head moment is %d",TotalHeadMoment);
    return 0;

}
```

amogh@Amogh: ~/Desktop/Practical Experiments

```
amogh@Amogh:~/Desktop/Practical Experiments$ vi fcfs_ds.c
amogh@Amogh:~/Desktop/Practical Experiments$ gcc -o fcfs_ds fcfs_ds.c
amogh@Amogh:~/Desktop/Practical Experiments$ ./ fcfs
bash: ./: Is a directory
amogh@Amogh:~/Desktop/Practical Experiments$ ./fcfs_ds
Enter the number of Requests
8
Enter the Requests sequence
95 180 34 119 11 123 62 64
Enter initial head position
50
amogh@Amogh:~/Desktop/Practical Experiments$
```

# EXPERIMENT-18 (SSTF DISK SCHEDULING)

```c
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int RQ[100],i,n,TotalHeadMoment=0,initial,count=0;
    printf("Enter the number of Requests\n");
    scanf("%d",&n);
    printf("Enter the Requests sequence\n");
    for(i=0;i<n;i++)
     scanf("%d",&RQ[i]);
    printf("Enter initial head position\n");
    scanf("%d",&initial);

    // logic for sstf disk scheduling

        /* loop will execute until all process is completed*/
    while(count!=n)
    {
        int min=1000,d,index;
        for(i=0;i<n;i++)
        {
           d=abs(RQ[i]-initial);
           if(min>d)
           {
               min=d;
               index=i;
           }

        }
        TotalHeadMoment=TotalHeadMoment+min;
        initial=RQ[index];
        // 1000 is for max
        // you can use any number
        RQ[index]=1000;
        count++;
    }

    printf("Total head movement is %d",TotalHeadMoment);
    return 0;
}
```

amogh@Amogh: ~/Desktop/Practical Experiments

```
amogh@Amogh:~/Desktop/Practical Experiments$ vi sstf_ds.c
amogh@Amogh:~/Desktop/Practical Experiments$ gcc -o sstf_ds sstf_ds.c
amogh@Amogh:~/Desktop/Practical Experiments$ ./sstf_ds
Enter the number of Requests
8
Enter the Requests sequence
95 180 34 119 11 123 62 64
Enter initial head position
50
amogh@Amogh:~/Desktop/Practical Experiments$
```

# EXPERIMENT-19 (C-SCAN)

```c
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
    printf("Enter the number of Requests\n");
    scanf("%d",&n);
    printf("Enter the Requests sequence\n");
    for(i=0;i<n;i++)
     scanf("%d",&RQ[i]);
    printf("Enter initial head position\n");
    scanf("%d",&initial);
    printf("Enter total disk size\n");
    scanf("%d",&size);
    printf("Enter the head movement direction for high 1 and for low 0\n");
    scanf("%d",&move);

    // logic for C-Scan disk scheduling

        /*logic for sort the request array */
    for(i=0;i<n;i++)
    {
        for( j=0;j<n-i-1;j++)
        {
            if(RQ[j]>RQ[j+1])
            {
                int temp;
                temp=RQ[j];
                RQ[j]=RQ[j+1];
                RQ[j+1]=temp;
            }

        }
    }

    int index;
    for(i=0;i<n;i++)
    {
        if(initial<RQ[i])
        {
            index=i;
            break;
        }
    }

        break;
    }
}

// if movement is towards high value
if(move==1)
{
    for(i=index;i<n;i++)
    {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
        initial=RQ[i];
    }
    //  last movement for max size
    TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
    /*movement max to min disk */
    TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
    initial=0;
    for( i=0;i<index;i++)
    {
         TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
         initial=RQ[i];

    }
}
// if movement is towards low value
else
{
    for(i=index-1;i>=0;i--)
    {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
        initial=RQ[i];
    }
    //  last movement for min size
    TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
    /*movement min to max disk */
    TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
    initial =size-1;
    for(i=n-1;i>=index;i--)
    {
         TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
         initial=RQ[i];

    }
}
```

amogh@Amogh: ~/Desktop/Practical Experiments

```
amogh@Amogh:~/Desktop/Practical Experiments$ vi cscan.c
amogh@Amogh:~/Desktop/Practical Experiments$ gcc -o cscan cscan.c
amogh@Amogh:~/Desktop/Practical Experiments$ ./cscan
Enter the number of Requests
8
Enter the Requests sequence
95 180 34 119 11 123 62 64
Enter initial head position
50
Enter total disk size
200
Enter the head movement direction for high 1 and for low 0
1
amogh@Amogh:~/Desktop/Practical Experiments$
```

# EXPERIMENT-20 (LOOK)

```c
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
    printf("Enter the number of Requests\n");
    scanf("%d",&n);
    printf("Enter the Requests sequence\n");
    for(i=0;i<n;i++)
     scanf("%d",&RQ[i]);
    printf("Enter initial head position\n");
    scanf("%d",&initial);
    printf("Enter total disk size\n");
    scanf("%d",&size);
    printf("Enter the head movement direction for high 1 and for low 0\n");
    scanf("%d",&move);

    // logic for look disk scheduling

        /*logic for sort the request array */
    for(i=0;i<n;i++)
    {
        for(j=0;j<n-i-1;j++)
        {
            if(RQ[j]>RQ[j+1])
            {
                int temp;
                temp=RQ[j];
                RQ[j]=RQ[j+1];
                RQ[j+1]=temp;
            }

        }
    }

    int index;
    for(i=0;i<n;i++)
    {
        if(initial<RQ[i])
        {
            index=i;
            break;
        }
    }

    for(i=0;i<n;i++)
    {
        if(initial<RQ[i])
        {
            index=i;
            break;
        }
    }

    // if movement is towards high value
    if(move==1)
    {
        for(i=index;i<n;i++)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];
        }

        for(i=index-1;i>=0;i--)
        {
             TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
             initial=RQ[i];

        }
    }
    // if movement is towards low value
    else
    {
        for(i=index-1;i>=0;i--)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];
        }

        for(i=index;i<n;i++)
        {
             TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
             initial=RQ[i];

        }
    }

    printf("Total head movement is %d",TotalHeadMoment);
    return 0;
}
```

amogh@Amogh: ~/Desktop/Practical Experiments

```
50
Enter total disk size
1
Enter the head movement direction for high 1 and for low 0
^C
amogh@Amogh:~/Desktop/Practical Experiments$ ./look
Enter the number of Requests
8
Enter the Requests sequence
95 180 34 119 11 123 62 64
Enter initial head position
50
Enter total disk size
200
Enter the head movement direction for high 1 and for low 0
1
amogh@Amogh:~/Desktop/Practical Experiments$
```