CSL 206

Operating Systems Lab Record

Midhun P Mohanan

S4 CSE B

Roll no: 30

BASIC LINUX COMMANDS

Sl.no	command	description	Synatx
1	pwd	display present working directory	pwd
2	mkdir	create a directory	mkdir dir_name
3	rmdir	delete a directory	rmdir dir_name
4	cd	change directory	cd
5	cd ~	change to user directory	cd~
6	cd ..	move to parent directory	cd ..
7	cd -	print directory	cd -
8	cat	to concatenate 2 text files	cat f1.txt f2.txt
9	cp	copy contents from one file to another	cp f1.txt f2.txt
10	clear	clear the terminal	clear
11	cmp	compare two text files	cmp f1.txt f2.txt
12	ls	list directory contents	ls
13	ls -al	list hidden and non hidden files	ls -al
14	ls -r	list non hidden files	ls -r
15	ls -a	list only hidden files	ls -a
16	ls -R	list subdirectories recursively	ls -R
17	ls -l	list files along with its details	ls -l
18	mv	move or rename files	mv
19	echo	print a textline provided	echo “text”
20	uname	display linux info	uname
21	who	shows currently logged in users	who
22	cal	show calendar	cal

23	history	show history of commands	history
24	date	display date and time	date
25	rm	delete files	rm file_name
26	sort	sort lines of text in a file	sort file_name
27	ping	check if a network is available	ping
28	reboot	reboot the system	reboot
29	man	display user manual of any command	man command
30	last	display list of last logged in users	last

Commands :

```
midhun@midhun:~/Desktop/test$ pwd
/home/midhun/Desktop/test
midhun@midhun:~/Desktop/test$ mkdir di
midhun@midhun:~/Desktop/test$ mkdir os
midhun@midhun:~/Desktop/test$ ls
di  os
midhun@midhun:~/Desktop/test$ rmdir os
midhun@midhun:~/Desktop/test$ ls
di
midhun@midhun:~/Desktop/test$ cd di
midhun@midhun:~/Desktop/test/di$ cat> f1.txt
hello
midhun@midhun:~/Desktop/test/di$ cat> f2.txt
fourteen
hi
midhun@midhun:~/Desktop/test/di$ sort f2.txt
hi
fourteen
midhun@midhun:~/Desktop/test/di$ cat f1.txt f2.txt
hello
fourteen
hi
midhun@midhun:~/Desktop/test/di$ cmp f1.txt f2.txt
f1.txt f2.txt differ: byte 1, line 1
midhun@midhun:~/Desktop/test/di$ ls -al
```

```
total 16
drwxrwxr-x 2 midhun midhun 4096 Oct  3 11:55 .
drwxrwxr-x 3 midhun midhun 4096 Oct  3 11:54 ..
-rw-rw-r-- 1 midhun midhun    6 Oct  3 11:55 f1.txt
-rw-rw-r-- 1 midhun midhun   17 Oct  3 11:55 f2.txt
midhun@midhun:~/Desktop/test/di$ ls -R
.:
f1.txt  f2.txt
midhun@midhun:~/Desktop/test/di$ echo fire
fire
midhun@midhun:~/Desktop/test/di$ uname
Linux
midhun@midhun:~/Desktop/test/di$ uname -o
GNU/Linux
midhun@midhun:~/Desktop/test/di$ who
midhun   :0           2021-10-03 10:05 (:0)
midhun@midhun:~/Desktop/test/di$ ping youtube.com
PING youtube.com (142.250.196.46) 56(84) bytes of data.
64 bytes from maa03s45-in-f14.1e100.net (142.250.196.46): icmp_seq=1 ttl=118 time=77.4 ms
64 bytes from maa03s45-in-f14.1e100.net (142.250.196.46): icmp_seq=2 ttl=118 time=22.3 ms
64 bytes from maa03s45-in-f14.1e100.net (142.250.196.46): icmp_seq=3 ttl=118 time=19.8 ms
^C
--- youtube.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 19.830/39.830/77.356/26.553 ms

midhun@midhun:~/Desktop/test/di$ date
Sunday 03 October 2021 10:50:19 PM IST
midhun@midhun:~/Desktop/test/di$ cal
    October 2021
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
midhun@midhun:~/Desktop$ last
midhun   :0           :0               Sun Oct  3 19:29   still logged in
reboot   system boot  5.11.0-34-generi Sun Oct  3 19:26   still running
midhun   :0           :0               Sun Oct  3 17:00 - crash  (02:25)
reboot   system boot  5.11.0-34-generi Sun Oct  3 16:50   still running
midhun   :0           :0               Sun Oct  3 10:05 - crash  (06:45)
reboot   system boot  5.11.0-34-generi Sun Oct  3 10:00   still running
midhun   :0           :0               Sat Oct  2 23:16 - down   (00:57)
reboot   system boot  5.11.0-34-generi Sat Oct  2 23:14 - 00:13  (00:58)
midhun   :0           :0               Sat Oct  2 16:31 - down   (00:47)
reboot   system boot  5.11.0-34-generi Sat Oct  2 16:29 - 17:18  (00:49)
```

```
midhun      :0          :0              Sat Oct  2 11:00 - down   (02:10)
reboot   system boot  5.11.0-34-generi Sat Oct  2 10:54 - 13:10  (02:16)
midhun      :0          :0              Fri Oct  1 21:47 - down   (00:56)
reboot   system boot  5.11.0-34-generi Fri Oct  1 21:44 - 22:44  (00:59)
midhun      :0          :0              Thu Sep 30 18:43 - down   (03:34)
reboot   system boot  5.11.0-34-generi Thu Sep 30 18:42 - 22:18  (03:35)
midhun      :0          :0              Wed Sep 29 20:09 - down   (03:26)
reboot   system boot  5.11.0-34-generi Wed Sep 29 20:02 - 23:35  (03:32)
midhun      :0          :0              Sat Sep 25 13:55 - down   (00:47)
reboot   system boot  5.11.0-34-generi Sat Sep 25 13:52 - 14:42  (00:50)
midhun      :0          :0              Tue Sep 21 22:40 - crash (3+15:11)
reboot   system boot  5.11.0-34-generi Tue Sep 21 22:39 - 14:42 (3+16:03)
midhun      :0          :0              Tue Sep 21 22:03 - down   (00:06)
reboot   system boot  5.11.0-34-generi Tue Sep 21 22:02 - 22:10  (00:07)
midhun      :0          :0              Sat Sep 18 16:16 - crash (3+05:46)


wtmp begins Tue Sep  7 10:55:21 2021

 midhun@midhun:~/Desktop/test/di$ history
 1193  cat lru.c
 1194  clear
 1195  cat fi.c
 1196  gcc fi.c
 1197  ./a.out
 1198  gcc fi.c
 1199  ./a.out
 1200  gedit fi.c
 1201  gcc fi.c
 1202  ./a.out
 1203  mv fi.c fitry.c
 1204  ls
 1205  rm lru.c
 1206  ls
 1207  #include<stdio.h>
 1208  ls
 1209  clear
 1210  exit
 1211  cd ..
 1212  cd ../..
 1213  ls
 1214  cd Desktop/
 1215  ls
 1216  history
 midhun@midhun:~/Desktop$
```

SHELL SCRIPT

1. Shell pgm to calculate factorial

Algorithm

```
1)  Start
2)  Define a function grt3
  2.1) if[$num1 –gt $num2]&&[$num1 –gt $num3]
       2.1.1)then print num1
  2.2) elif [ $num2 –gt $num1] && [$num2 –gt $num3]
       2.2.1) then print num2
  2.3)else
     2.3.1) print num3
  2.4) fi
3)  Read num1 num2 num3
4)  Print greatest of three numbers is:
5)  Call function Grt3 num1, num2, num3
6)  stop
```

Code

```
fact()
 {
 fact=1
 for ((i=2;i<=num;i++))
 {
 fact=$((fact*i))

 }
 echo "factorial of $num is: $fact"
 }
echo " Enter a number"
read num
fact num
```

OUTPUT

```
midhun@midhun:~/Desktop/os/pgm/shellpgms$ ./fact.sh
 Enter a number
6
factorial of 6 is: 720
```

2.Shell pgm to find greatest of three numbers

Algorithm

```
1)      Start
2)      Define a function grt3
2.1) if[$num1 –gt $num2]&&[$num1 –gt $num3]
        2.1.1)then print num1
2.2) elif [ $num2 –gt $num1] && [$num2 –gt $num3]
       2.2.1) then print num2
2.3)else
     2.3.1) print num3
2.4) fi

3)      Read num1 num2 num3
4)      Print greatest of three numbers is:
5)      Call function Grt3 num1, num2, num3
6)      stop
```

Code

```
grt3()
 {
 if [ $num1 -gt $num2 ] && [ $num1 -gt $num3 ]
 then
 echo $num1
 elif [ $num2 -gt $num1 ] && [ $num2 -gt $num3 ]
 then
 echo $num2
 else
 echo $num3
 fi
 }
 echo "Enter Num1"
 read num1
 echo "Enter Num2"
 read num2
 echo "Enter Num3"
 read num3
 echo " the greatest of three is: "
 grt3 num1 num2 num3
```

OUTPUT

```
midhun@midhun:~/Desktop/os/pgm/shellpgms$ ./great.sh
Enter Num1
4
Enter Num2
7
```

```
Enter Num3
2
 the greatest of three is:
7
```

3.Shell pgm to find sum and average

Algorithm

```
1)      Start
2)      Read size
3)      I=1 , sum=0
4)      Read numbers
5)      While [ $i –le $ n]
6)      Do
  6.1) read num
  6.2)  sum = $((sum+num))
  6.3) i=$((i+1))
  6.4) done

7) avg =$( echo $sum / $n | bc-l)

8) print sum and avg

 9) stop
```

Code

```
echo " enter the size"
read n
i=1
sum=0
echo " enter the numbers"
while [ $i -le $n ]
do
 read num
 sum=$((sum + num))
 i=$((i+1))
 done
avg=$( echo $sum / $n | bc -l)
echo “Sum : $sum”
echo " average is :$avg"
```

OUTPUT

```
midhun@midhun:~/Desktop/os/pgm/shellpgms$ ./avg.sh
 enter the size
```

```
3
 enter the numbers
4
8
3
 Sum: 15
 Average :5.00000000000000000000
```

5.Shell pgm to show various System configurations

```
echo "Current Shell :echo $SHELL"
echo "Current Directory: $PWD"
echo "Home Directory: $HOME"
echo "OS type : $OSTYPE"
echo "currently logged users "
who --count
echo "current path : $PATH"
```

OUTPUT

```
midhun@midhun:~/Desktop/os/pgm/shellpgms$ ./config.sh
Current Shell :echo /bin/bash
Current Directory: /home/midhun/Desktop/os/pgm/shellpgms
Home Directory: /home/midhun
OS type : linux-gnu
currently logged users
midhun
# users=1
current path :
        /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
```

7.Shell script for menu driven calculator

Algorithm

```
1.  Start
2.  Input numbers a&b
3.  Assign ch=0
4.  while (ch!=5)
        4.1.  print options
        4.2.  read choice
        4.3.  if case=1
            4.3.1.  result = a+b
        4.4.  if case=2
            4.4.1.  result = a-b
            4.5.    if case=3
                4.5.1.    result = a*b
            4.6.    if case=4
                4.6.1.    result=a/b
            4.7.    if case=5
                4.7.1.    exit
5.    stop
```

Code

```
echo "SIMPLE CALCULATOR"
res=0
ch=0
echo "enter number A:"
read a
echo "enter number B:"
read b
while [ $ch -ne 5 ]
do
echo "1.Addition"
echo "2.Subtraction"
echo "3.Multiplication"
echo "4.Division"
echo "5.Exit"
echo "enter choice"
read ch
case $ch in
1)res=$(echo " $a + $b" | bc -l)
echo "Sum: " $res ;;
2)res=$(echo " $a - $b" | bc -l)
echo "Difference = " $res ;;
3)res=$(echo " $a * $b" | bc -l)
echo "Product = " $res ;;
4)res=$(echo " $a / $b" | bc -l)
echo "Quotient" $res ;;
5)exit;;
*)echo "invalid choice"
esac
done
```

OUTPUT

```
midhun@midhun:~/Desktop/os/pgm/shellpgms$ ./t.sh
SIMPLE CALCULATOR
enter number A:
3
enter number B:
4
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Exit
enter choice
1
Sum:  7
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Exit
enter choice
2
Difference =  -1
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Exit
enter choice
3
Product = 12
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Exit
enter choice
5
```

System Calls

1.Write a C program to create a child process that lists the files and directories and the parent process waits till the child completes. Also print the PID’s of parent and child process.

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>
int main()
{
pid_t p1;
p1=fork();
if(p1<0)
{
        printf("failed");
        return 1;
}
else if(p1==0)
{

        execlp("/bin/ls","ls",NULL);
        printf("pid of child =%d",getpid());
        printf("\n");


}

else
{
        wait(NULL);
        printf("Child pid: %d \n",getpid());

        printf("pid of parent =%d\n",getppid());
}
return 0;
}
```

OUTPUT

```
midhun@midhun:~/Desktop/os/pgm$ ./child
a.out  child.c    fcfs           shared1    shared2    shell  sjf.c
child  factorial  fcfsprocess.c  shared1.c  shared2.c  sj     sjfn.c
Child pid: 11516
pid of parent =2783
```

2. Create 5 child process and print pid of child

```
#include <stdio.h>
#include <sys/wait.h>
#include<sys/types.h>
#include <unistd.h>
#include <stdlib.h>

int main()
{
   for (int i = 0; i < 5; i++)
   {
      if (fork() == 0)
      {
         printf("Child PID :%d created Parent PID: %d\n", getpid(), getppid());
         exit(0);
      }
   }
   for (int i = 0; i < 5; i++)
   {
      wait(NULL);
   }
   return 0;
}
```

OUTPUT

```
midhun@midhun:~/Desktop/os/pgm$ ./a.out
Child PID :11682 created Parent PID: 11681
Child PID :11684 created Parent PID: 11681
Child PID :11685 created Parent PID: 11681
Child PID :11683 created Parent PID: 11681
Child PID :11686 created Parent PID: 11681
```

3.C program using system calls stat, opendir, closedir to display the files and their sizes.

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<stdlib.h>
#include<dirent.h>

int main(int argc, char **argv){
          struct stat buf;
          int exists;
          DIR *d;
          struct dirent *de;
```

```
    d=opendir("/home/midhun/Desktop/os/pgm/shellpgms");
    if(d==NULL)
        printf("File not found");
    for(de=readdir(d);de!=NULL;de=readdir(d))
    {
        exists=stat(de->d_name,&buf);
        if(exists<0)
            printf("Error");
        else
            printf("%s %lld\n",de->d_name,buf.st_size);
    }
    closedir(d);
    return 0;
}
```

OUTPUT

```
midhun@midhun:~/Desktop/os/pgm/shellpgms$ ./a.out
greatest.sh 315
test.sh 528
.. 4096
configurations.sh 191
average.sh 208
c.sh 593
factorial.sh 148
. 4096
a.out 16984
```

IPC using Shared Memory

1.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>
int main()
{
        int i;
        void *shared_memory;
        char buff[100];
        int shmid;
        shmid=shmget((key_t)2345, 1024, 0666|IPC_CREAT);
        printf("\nKey of shared memory is %d\n",shmid);
        shared_memory=shmat(shmid,NULL,0);
        printf("Process attached at %p\n",shared_memory);
        printf("Enter some data to write to shared memory: \n");
        read(0,buff,100);
        strcpy(shared_memory,buff);
        printf("\nEntered data : %s\n",(char *)shared_memory);
}
```

2.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>
int main()
{
        int i;
        void *shared_memory;
        char buff[100];
        int shmid;
        shmid=shmget((key_t)2345, 1024, 0666);
        printf("\nKey of shared memory is %d\n",shmid);
        shared_memory=shmat(shmid,NULL,0);
        printf("Process attached at %p\n",shared_memory);
        printf("Data from shared memory : %s\n",(char *)shared_memory);
}
```

OUTPUT

```
midhun@midhun:~/Desktop/os/pgm$ ./shared1
Enter some data to write to shared memory:
there
Key of shared memory is 98344
Process attached at 0x7fe2227b8000

Entered data : there

midhun@midhun:~/Desktop/os/pgm$ ./shared2

Key of shared memory is 98344
Process attached at 0x7f8998957000
Data from shared memory : there
```

Process Scheduling

FCFS

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

struct process{
  int at,bt,ct,tt,wt;
  char name[50];
}p[20],temp;

int main()
{
  int n,k=0,g_time=0,time_taken=0;
  float sum_tt=0.0,sum_wt=0.0;
  printf("\n\tFCFS\n");
  printf("\nEnter the number of processes : ");
  scanf("%d", &n);
  for(int i=0;i<n;i++){
    printf("\nEnter the name of the process %d: ", (i+1));
    __fpurge(stdin);
    fgets(p[i].name, 20, stdin);
    printf("   Enter the arrival time of the process : ");
    scanf("%d", &p[i].at);
    printf("   Enter the burst time of the process : ");
    scanf("%d", &p[i].bt);
    p[i].name[strcspn(p[i].name, "\n")] = 0;
  }

  for(int i=0;i<n;i++){
    for(int j=i+1;j<n;j++){
      if(p[i].at>p[j].at){
        temp = p[i];
        p[i] = p[j];
        p[j] = temp;
      }
    }
  }

  int i = 0,j = 0;
  printf("\nPROCESS TABLE");
  printf("\n Process Name\t| Process AT\t| Process BT\t| Process CT\t| Process TT\t| Process
           WT\n");

  while(i<n){
    if(time_taken >= p[i].at){
```

```
   time_taken += p[i].bt;
   p[i].ct = time_taken;
   p[i].tt = p[i].ct - p[i].at;
   p[i].wt = time_taken - p[i].bt - p[i].at;
   sum_tt += p[i].tt;
   sum_wt += p[i].wt;
   printf("\n\t %s\t \t %d\t \t %d\t \t %d\t \t %d\t \t %d",p[i].name, p[i].at, p[i].bt, p[i].ct,
       p[i].tt, p[i].wt);
   i++;
  }else{
   time_taken = p[i].at;
  }
 }

 printf("\n\nGantt Chart ");
 printf("\n| 0 |");
 while(k<n){
  if(p[k].at<=g_time){
   printf("    %s    | %d |", p[k].name, p[k].ct);
   g_time += p[k].bt;
   k++;
  }else{
   g_time = p[k].at;
   printf("    idle    | %d |", g_time);
  }
 }

 printf("\n\n  Average Waiting Time : %f",(sum_wt/n));
 printf("\n  Average Turnaround Time : %f\n",(sum_tt/n));

}
```

OUTPUT

```
midhun@midhun:~/Desktop/os$ ./a.out

        FCFS
Enter the number of processes : 4

Enter the name of the process 1: a
  Enter the arrival time of the process : 0
  Enter the burst time of the process : 8

Enter the name of the process 2: b
  Enter the arrival time of the process : 1
```

```
   Enter the burst time of the process : 4

Enter the name of the process 3: c
   Enter the arrival time of the process : 2
   Enter the burst time of the process : 1

Enter the name of the process 4: d
   Enter the arrival time of the process : 3
   Enter the burst time of the process : 5

PROCESS TABLE
 Process Name         | Process AT    | Process BT    | Process CT    | Process TT    | Process WT

            a            0              8              8              8              0
            b            1              4              12             11             7
            c            2              1              13             11             10
            d            3              5              18             15             10


 Gantt Chart
| 0 |    a    | 8 |    b    | 12 |    c    | 13 |    d    | 18 |

  Average Waiting Time : 6.750000
  Average Turnaround Time : 11.250000
```

Shortest Job First Scheduling

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

struct process{
  int at,bt,ct,tt,wt;
  char name[50];
}p[20],temp;

int main()
{
  int n,k=0,g_time=0,time=0;
  float sum_tt=0.0,sum_wt=0.0;
  printf("\n\t SJF \n");
  printf("\nEnter the number of process : ");
  scanf("%d", &n);
  for(int i=0;i<n;i++){
    printf("\nEnter the name of the process %d: ", (i+1));
    __fpurge(stdin);
    fgets(p[i].name, 20, stdin);
    printf("   Enter the arrival time of the process : ");
    scanf("%d", &p[i].at);
    printf("   Enter the burst time of the process : ");
    scanf("%d", &p[i].bt);
    p[i].name[strcspn(p[i].name, "\n")] = 0;
  }

  for(int i=0;i<n;i++){
    for(int j=i+1;j<n;j++){
      if(p[i].at>p[j].at){
        temp = p[i];
        p[i] = p[j];
        p[j] = temp;
      }
    }
  }

  int i = 0,j = 0;
  printf("\n Process Table ");
  printf("\nProcess Name \t| Process AT \t| Process BT \t| Process CT \t| Process TT \t| Process
              WT\n");

  while(i<n){
    if(time >= p[i].at){

      j = i+1;
      while(j<n){
        if(p[j].bt < p[i].bt && p[j].at<=time){
          temp = p[j];
          p[j] = p[i];
          p[i] = temp;
        }
        j++;
      }

      time += p[i].bt;
      p[i].ct = time;
      p[i].tt = p[i].ct - p[i].at;
      p[i].wt = time - p[i].bt - p[i].at;
      sum_tt += p[i].tt;
      sum_wt += p[i].wt;
      printf("\n%s \t\t  %d \t\t  %d \t\t  %d \t\t  %d \t\t  %d",p[i].name, p[i].at, p[i].bt, p[i].ct, p[i].tt,
              p[i].wt);
      i++;
    }else{
      time = p[i].at;
    }
  }

  printf("\n\nGantt Chart");
  printf("\n| 0 |");
```

```
  while(k<n){
    if(p[k].at<=g_time){
      printf("    %s    | %d |", p[k].name, p[k].ct);
      g_time += p[k].bt;
      k++;
    }else{
      g_time = p[k].at;
      printf("    idle    | %d |", g_time);
    }
  }

  printf("\n\nAverage Waiting Time : %f",(sum_wt/n));
  printf("\nAverage Turnaround Time : %f\n",(sum_tt/n));

}
```

OUTPUT

```
midhun@midhun:~/Desktop/os$ ./sjf

            SJF Scheduling
Enter the number of process : 4

Enter the name of the process 1: a
   Enter the arrival time of the process : 0
   Enter the burst time of the process : 8

Enter the name of the process 2: b
   Enter the arrival time of the process : 0
   Enter the burst time of the process : 4

Enter the name of the process 3: c
   Enter the arrival time of the process : 2
   Enter the burst time of the process : 3

Enter the name of the process 4: d
   Enter the arrival time of the process : 3
   Enter the burst time of the process : 4

Process Table
Process Name | Process AT    | Process BT    | Process CT    | Process TT    | Process WT

b                  0              4              4              4              0
c                  2              3              7              5              2
d                  3              4              11             8              4
a                  0              8              19             19             11


 Gantt Chart
| 0 |    b    | 4 |    c    | 7 |    d    | 11 |    a    | 19 |
Average Waiting Time : 4.250000
Average Turnaround Time : 9.000000
```

Priority Scheduling

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

struct process{
  int at,bt,ct,tt,wt,pt;
  char name[50];
}p[20],temp;

int main()
{
  int n,k=0,g_time=0,time_taken=0;
  float sum_tt=0.0,sum_wt=0.0;
  printf("\n PRIORITY - Non Preemptive\n");
  printf("\nEnter the number of process : ");
  scanf("%d", &n);
  for(int i=0;i<n;i++){
    printf("\nEnter the name of the process %d: ", (i+1));
    __fpurge(stdin);
    fgets(p[i].name, 20, stdin);
    printf("   Enter the arrival time of the process : ");
    scanf("%d", &p[i].at);
    printf("   Enter the burst time of the process : ");
    scanf("%d", &p[i].bt);
    printf("   Enter the priority of the process : ");
    scanf("%d", &p[i].pt);
    p[i].name[strcspn(p[i].name, "\n")] = 0;
  }

  for(int i=0;i<n;i++){
    for(int j=i+1;j<n;j++){
      if(p[i].at>p[j].at){
        temp = p[i];
        p[i] = p[j];
        p[j] = temp;
      }
    }
  }

  int i = 0,j = 0;
  printf("Process Table ");
  printf("\nProcess Name \t| Process AT \t| Process BT \t| Process CT \t| Process TT \t| Process WT \t|
            Priority\n");
```

```
 while(i<n){
  if(time_taken >= p[i].at){
   j = i+1;
   while(j<n){
    if(p[j].pt < p[i].pt && p[j].at<=time_taken){
     temp = p[j];
     p[j] = p[i];
     p[i] = temp;
    }
    j++;
   }
   time_taken += p[i].bt;
   p[i].ct = time_taken;
   p[i].tt = p[i].ct - p[i].at;
   p[i].wt = time_taken - p[i].bt - p[i].at;
   sum_tt += p[i].tt;
   sum_wt += p[i].wt;
   printf("\n%s \t\t  %d \t\t  %d \t\t  %d \t\t  %d \t\t  %d \t\t  %d",p[i].name, p[i].at, p[i].bt, p[i].ct,
       p[i].tt, p[i].wt, p[i].pt);
   i++;
  }else{
   time_taken = p[i].at;
  }
 }
 printf("\n Gantt Chart ");
 printf("\n| 0 |");
 while(k<n){
  if(p[k].at<=g_time){
   printf("    %s    | %d |", p[k].name, p[k].ct);
   g_time += p[k].bt;
   k++;
  }else{
   g_time = p[k].at;
   printf("    idle    | %d |", g_time);
  }
 }
 printf("\n\nAverage Waiting Time : %f",(sum_wt/n));
 printf("\nAverage Turnaround Time : %f\n",(sum_tt/n));
}
```

OUTPUT

```
midhun@midhun:~/Desktop/os$ ./priority
 PRIORITY - Non Preemptive

Enter the number of process : 4

Enter the name of the process 1: a
   Enter the arrival time of the process : 0
   Enter the burst time of the process : 8
```

```
  Enter the priority of the process : 5

Enter the name of the process 2: b
  Enter the arrival time of the process : 1
  Enter the burst time of the process : 4
  Enter the priority of the process : 2

Enter the name of the process 3: c
  Enter the arrival time of the process : 2
  Enter the burst time of the process : 1
  Enter the priority of the process : 1

Enter the name of the process 4: d
  Enter the arrival time of the process : 3
  Enter the burst time of the process : 5
  Enter the priority of the process : 3

Process Table

Process Name  | Process AT  | Process BT  | Process CT  | Process TT  | Process WT  | Priority

a             0             8             8             8             0             5
c             2             1             9             7             6             1
b             1             4             13            12            8             2
d             3             5             18            15            10            3

Gantt Chart
| 0 |   a   | 8 |   c   | 9 |   b   | 13 |   d   | 18 |

Average Waiting Time : 6.000000
Average Turnaround Time : 10.500000
```

Round Robin Scheduling

```
#include<stdio.h>
#include<string.h>
#define MAX 20

int frontR = -1,rearR = -1,top_done = 0;
struct process
{
	char pname[10];
	int at,bt,wt,tt,error,index;
}p[MAX],temp,ready_queue[MAX];

struct done
{
	char pname[10];
	int time;
```

```
}done_stack[MAX];
void insert_readyQueue(struct process k)
{
    if(rearR == -1 && frontR == -1)
        frontR = 0;
    if(rearR <= MAX)
    {
        rearR++;
        if(k.error != 0)
            k.error = 0;
        ready_queue[rearR] = k;
    }
}
struct process pop_readyQueue()
{
    struct process p;
    if(frontR != -1 && frontR <= rearR)
    {
        p = ready_queue[frontR];
        frontR ++;

        if(frontR > rearR)
        {
            frontR = -1;
            rearR = -1;
        }
    }
    else
        p.error = 1;
        return p;
    }
void insert_doneStack(char* pname,int time)
{
    struct done m;
    strncpy(m.pname, pname, 10);
    m.time = time;
    done_stack[top_done] = m;
    top_done++;
}

int main() {

    int i,j,n,time_elapsed = 0.0,time_slice;
    float sumWT = 0.0,sumTT = 0.0;


  printf("\n Round Robin Scheduling \n");
    printf("Enter the number of process : ");
    scanf("%d",&n);
```

```
for(i=0;i<n;i++)
{
printf("\nEnter the process name : ");
__fpurge(stdin);
fgets(p[i].pname,MAX,stdin);

p[i].pname[strcspn(p[i].pname, "\n")] = 0;
printf("Enter the arrival time : ");
scanf("%d",&p[i].at);
printf("Enter the burst time : ");

scanf("%d",&p[i].bt);
}
printf("\nEnter the time slice : ");
scanf("%d",&time_slice);

for(i=0;i<n;i++)
{
for(j=i+1;j<n;j++)
{
        if(p[i].at > p[j].at)
        {
                temp=p[i];
                p[i] = p[j];
                p[j] = temp;
        }
}
        p[i].index = i;
}


temp = p[0];
i=1;
while(temp.error == 0 )
{
        if(time_elapsed >= temp.at)
        {
                insert_doneStack(temp.pname,time_elapsed);

                if(temp.bt < time_slice)
                {
                        time_elapsed += temp.bt;
                        temp.bt =0;
                }
                else
                {
                        time_elapsed +=time_slice;
                        temp.bt -= time_slice;
                }
```

```
            if(temp.bt == 0)
            {

                temp.tt = time_elapsed - temp.at;
                temp.wt = temp.tt - p[temp.index].bt;
                p[temp.index].tt = temp.tt;
                p[temp.index].wt = temp.wt;
                sumWT += temp.wt;

                sumTT += temp.tt;

            }

            if(i < n)
            {
                j= p[i].at;
                while(j <= time_elapsed)
                {
                    insert_readyQueue(p[i]);
                    i++;
                    if(i < n)
                        j = p[i].at;
                    else
                        break;

                }
            }


            if(temp.bt !=0 )
                insert_readyQueue(temp);
            temp = pop_readyQueue();
            if(temp.error !=0 && i< n )
            {
                temp = p[i];
                i++;
            }
        }
        else
        {
            char arr[] = "idle";
            insert_doneStack(arr,time_elapsed);
            time_elapsed += temp.at - time_elapsed;
        }
    }
printf("\n\n\nProcess Table \n\n");
    printf("\n| Name\t| Arrival time\t| Burst time\t| Waiting time\t| Turnaround");
    for(i=0;i<n;i++)
    {
```

```
                printf("\n\n| %2s\t| %2d\t\t| %2d\t\t| %2d\t\t|
        %2d",p[i].pname,p[i].at,p[i].bt,p[i].wt,p[i].tt);
        }
        //Gantt chart
        printf("\n\n Gantt Chart \n\n");

        for(i=0;i<top_done;i++)
        {
                printf("|%4s\t",done_stack[i].pname);
        }
        printf("|\t");
        printf("\n");
        for(i=0;i<top_done;i++)
                printf("%d\t", done_stack[i].time);
        printf("%d\t",time_elapsed);
        printf("\n\nAverage Turn Around Time : %f",sumTT/n);
        printf("\nAverage Waiting Time : %f\n",sumWT/n);
        return 0;
}
```

OUTPUT

```
midhun@midhun:~/Desktop/os$ ./round

 Round Robin Scheduling
Enter the number of process : 4

Enter the process name : a
Enter the arrival time : 0
Enter the burst time : 8

Enter the process name : b
Enter the arrival time : 1
Enter the burst time : 4

Enter the process name : c
Enter the arrival time : 2
Enter the burst time : 1

Enter the process name : d
Enter the arrival time : 3
Enter the burst time : 5

Enter the time slice : 2
```

```
Process Table

| Name         | Arrival time | Burst time  | Waiting time     | Turnaround
|  a           | 0            | 8           | 9                | 17
|  b           | 1            | 4           | 6                | 10
|  c           | 2            | 1           | 2                |  3
|  d           | 3            | 5           | 10               | 15

 Gantt Chart

|  a    |  b    |  c    |  a    |  d    |  b    |  a    |  d    |  a    |  d    |
0       2       4       5       7       9       11      13      15      17      18

Average Turnaround Time : 11.250000
Average Waiting Time : 6.750000
```

Round Robin Scheduling

```
#include<stdio.h>
#include<string.h>
#define MAX 20

int frontR = -1,rearR = -1,top_done = 0;
struct process
{
        char pname[10];
        int at,bt,wt,tt,error,index;
}p[MAX],temp,ready_queue[MAX];

struct done
{
        char pname[10];
        int time;

}done_stack[MAX];
void insert_readyQueue(struct process k)
{
        if(rearR == -1 && frontR == -1)
                frontR = 0;
        if(rearR <= MAX)
        {
                rearR++;
                if(k.error != 0)
                        k.error = 0;
                ready_queue[rearR] = k;
        }
```

```
}
struct process pop_readyQueue()
{
        struct process p;
        if(frontR != -1 && frontR <= rearR)
        {
                p = ready_queue[frontR];
                frontR ++;

                if(frontR > rearR)
                {
                        frontR = -1;
                        rearR = -1;
                }
        }
        else
                p.error = 1;
                return p;
        }
void insert_doneStack(char* pname,int time)
{
        struct done m;
        strncpy(m.pname, pname, 10);
        m.time = time;
        done_stack[top_done] = m;
        top_done++;
}

int main() {

        int i,j,n,time_elapsed = 0.0,time_slice;
        float sumWT = 0.0,sumTT = 0.0;


      printf("\n Round Robin Scheduling \n");
        printf("Enter the number of process : ");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
        printf("\nEnter the process name : ");
        __fpurge(stdin);
        fgets(p[i].pname,MAX,stdin);

        p[i].pname[strcspn(p[i].pname, "\n")] = 0;
        printf("Enter the arrival time : ");
        scanf("%d",&p[i].at);
        printf("Enter the burst time : ");

        scanf("%d",&p[i].bt);
        }
```

```
printf("\nEnter the time slice : ");
scanf("%d",&time_slice);

for(i=0;i<n;i++)
{
for(j=i+1;j<n;j++)
{
    if(p[i].at > p[j].at)
    {
        temp=p[i];
        p[i] = p[j];
        p[j] = temp;
    }
}
    p[i].index = i;
}


temp = p[0];
i=1;
while(temp.error == 0 )
{
    if(time_elapsed >= temp.at)
    {
        insert_doneStack(temp.pname,time_elapsed);

        if(temp.bt < time_slice)
        {
            time_elapsed += temp.bt;
            temp.bt =0;
        }
        else
        {
            time_elapsed +=time_slice;
            temp.bt -= time_slice;
        }

        if(temp.bt == 0)
        {

            temp.tt = time_elapsed - temp.at;
            temp.wt = temp.tt - p[temp.index].bt;
            p[temp.index].tt = temp.tt;
            p[temp.index].wt = temp.wt;
            sumWT += temp.wt;

            sumTT += temp.tt;

        }
```

```
            if(i < n)
            {
                j= p[i].at;
                while(j <= time_elapsed)
                {
                    insert_readyQueue(p[i]);
                    i++;
                    if(i < n)
                        j = p[i].at;
                    else
                        break;

                }
            }


            if(temp.bt !=0 )
                insert_readyQueue(temp);
            temp = pop_readyQueue();
            if(temp.error !=0 && i< n )
            {
                temp = p[i];
                i++;
            }
        }
        else
        {
            char arr[] = "idle";
            insert_doneStack(arr,time_elapsed);
            time_elapsed += temp.at - time_elapsed;
        }
    }
printf("\n\n\nProcess Table \n\n");
    printf("\n| Name\t| Arrival time\t| Burst time\t| Waiting time\t| Turnaround time");
    for(i=0;i<n;i++)
    {
        printf("\n\n| %2s\t| %2d\t\t| %2d\t\t| %2d\t\t| %2d",p[i].pname,p[i].at,p[i].bt,p[i].wt,p[i].tt);
    }
    //Gantt chart
    printf("\n\n\n Gantt Chart \n\n");

    for(i=0;i<top_done;i++)
    {
        printf("|%4s\t",done_stack[i].pname);
    }
    printf("|\t");
    printf("\n");
    for(i=0;i<top_done;i++)
        printf("%d\t", done_stack[i].time);
    printf("%d\t",time_elapsed);
```

```
        printf("\n\nAverage Turn Around Time : %f",sumTT/n);
        printf("\nAverage Waiting Time : %f\n",sumWT/n);
        return 0;
}
```

OUTPUT

```
midhun@midhun:~/Desktop/os/aoutput$ ./round
Round Robin Scheduling
Enter the number of process : 3

Enter the process name : a
Enter the arrival time : 4
Enter the burst time : 6

Enter the process name : b
Enter the arrival time : 2
Enter the burst time : 8

Enter the process name : c
Enter the arrival time : 8
Enter the burst time : 3

Enter the time slice : 2

Process Table
| Name      | Arrival time  | Burst time  | Waiting time      | Turnaround time

|  b        |  2            |  8          |  9                | 17
|  a        |  4            |  6          |  6                | 12
|  c        |  8            |  3          |  6                |  9

 Gantt Chart

|idle  |  b   |  a   |  b   |  a   |  c   |  b   |  a   |  c   |  b   |
0      2      4      6      8      10     12     14     16     17     19

Average Turn Around Time : 12.666667
Average Waiting Time : 7.000000
```

Producer Consumer Problem

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

sem_t mutex,empty,full;
int buffer[5],get=0,item=0,gitem,put=0,pro[20],con[20];

void *producer(void *args)
{
    do
    {
        sem_wait(&empty);
        sem_wait(&mutex);
        buffer[put%5]=item;
        item++;
        printf("producer %d produces %d item buffered[%d] : %d\n",(*(int
    *)args),buffer[put%5],put%5,item);
        put++;
        sem_post(&mutex);
        sem_post(&full);
        sleep(3);
    }while(1);
}

void *consumer(void *args)
{
    do
    {
        sem_wait(&full);
        sem_wait(&mutex);
        gitem=buffer[get%5];
        printf("consumer %d consumes %d item buffered[%d] : %d\n",(*(int
    *)args),gitem,get%5,gitem);
        get++;
        sem_post(&mutex);
        sem_post(&full);
        sleep(2);
    }while(1);
}

void main()
{
    int p,c,k,j;
    pthread_t a[10],b[10];
    sem_init(&mutex,0,1);
    sem_init(&full,0,0);
    sem_init(&empty,0,5);
```

```
        printf("Enter number of Producers : ");
        scanf("%d",&p);
        printf("Enter number of Consumers:");
        scanf("%d",&c);
        for(j=0;j<p;j++)
        {
                pro[j]=j;
                pthread_create(&a[j],NULL,producer,&pro[j]);
        }
        for(k=0;k<c;k++)
        {
                con[k]=k;
                pthread_create(&b[k],NULL,consumer,&con[k]);
        }
        for(j=0;j<p;j++)
        {
                pthread_join(&a[j],NULL);
        }
        for(k=0;k<c;k++)
        {
                pthread_join(&b[k],NULL);
        }
}
```

OUTPUT

```
midhun@midhun:~/Desktop/os/aoutput$ ./pro
Enter number of Producers : 4
Enter number of Consumers:4
producer 3 produces 0 item buffered[0] : 1
consumer 0 consumes 0 item buffered[0] : 0
producer 1 produces 1 item buffered[1] : 2
producer 2 produces 2 item buffered[2] : 3
producer 0 produces 3 item buffered[3] : 4
consumer 1 consumes 1 item buffered[1] : 1
consumer 2 consumes 2 item buffered[2] : 2
consumer 3 consumes 3 item buffered[3] : 3
consumer 0 consumes 0 item buffered[4] : 0
consumer 1 consumes 0 item buffered[0] : 0
consumer 2 consumes 1 item buffered[1] : 1
consumer 3 consumes 2 item buffered[2] : 2
```

PAGE REPLACEMENT

FIFO

```
#include<stdio.h>
int i,j,fno,pno,flag=0,ref[50],frm[50],fault=0,victim=-1;
void main()
{
        printf("\n\nFIFO Page Replacement\n");
        printf("\nEnter no. of frames    : ");
        scanf("%d",&fno);
        printf("\nEnter no. of pages     : ");
        scanf("%d",&pno);

        printf("\nEnter the page numbers : ");
        for(i=0;i<pno;i++)
                scanf("%d",&ref[i]);
        for(i=1;i<=fno;i++)
                frm[i]=-1;
        printf("\n");
        for(i=0;i<pno;i++)
        {
                flag=0;
                printf("\nPage No. %d : ",ref[i]);
                for(j=0;j<pno;j++)
                {
                 if(frm[j]==ref[i])
                 { flag=1;break; }
                }
                if(flag==0)
                {
                  fault++; victim++;
                  victim=victim%fno;
                  frm[victim]=ref[i];
                  for(j=0;j<fno;j++)
                    printf("%4d",frm[j]);
                }
        }
printf("\n\nNo. of Page Faults = %d\n\n",fault);
}
```

OUTPUT

```
midhun@midhun:~/Desktop/os$ ./fifo

FIFO Page Replacement
Enter no. of frames    : 3
Enter no. of pages     : 7
Enter the page numbers : 1 2 5 3 7 5 9

Page No. 1 :    1  -1  -1
Page No. 2 :    1   2  -1
Page No. 5 :    1   2   5
Page No. 3 :    3   2   5
Page No. 7 :    3   7   5
Page No. 5 :
Page No. 9 :    3   7   9

No. of Page Faults = 6
```

LRU

```
#include<stdio.h>
void main(){
   int i,j,k,min,page[25],set[10],count[10],flag[25],n,f,pf=0,next=1;
   printf("Enter the number of frames : ");
   scanf("%d",&f);
   for(i=0;i<f;i++){
         count[i]=0;
         set[i]=-1;
   }
   printf("Enter number of pages: ");
   scanf("%d",&n);
   printf("Enter the page numbers: ");
   for(i=0;i<n;i++){
         scanf("%d",&page[i]);
         flag[i]=0;
   }

   printf("\nPage replacing.. \n");
   for(i=0;i<n;i++)
   {
         printf("%d ->\t",page[i]);
         for(j=0;j<f;j++){
                  if(set[j]==page[i]){
```

```
                flag[i]=1;
                count[j]=next;
                next++;
            }
        }
        if(flag[i]==0){
            if(i<f){
                set[i]=page[i];
                count[i]=next;
                next++;
            }
            else{
                min=0;
                for(j=1;j<f;j++)
                if(count[min]>count[j])
                min=j;
                set[min]=page[i];
                count[min]=next;
                next++;
            }
            pf++;
        }
        for(j=0;j<f;j++)
        printf("%2d\t",set[j]);
        if(flag[i]==0)
        printf("PF No. : %d",pf);
        printf("\n");
    }
    printf("\nTotal page faults: %d\n",pf);
}
```

OUTPUT

```
midhun@midhun:~/Desktop/os$ ./lru
Enter the number of frames : 3
Enter number of pages: 7
Enter the page numbers: 3 5 8 4 4 2 9

Page replacing..
3 ->    3      -1      -1      PF No. : 1
5 ->    3       5      -1      PF No. : 2
8 ->    3       5       8      PF No. : 3
4 ->    4       5       8      PF No. : 4
```

```
4 ->    4       5       8
2 ->    4       2       8       PF No. : 5
9 ->    4       2       9       PF No. : 6

Total page faults: 6
```

DISK SCHEDULING

FCFS

```
#include<stdio.h>
void main()
{
        int t[25],n,i,move,t_move=0;
        float avg_move;
        printf("Enter current head position: ");
        scanf("%d",&t[0]);
        printf("Enter number of tracks: ");
        scanf("%d",&n);
        printf("Enter tracks to be traversed: ");
        for(i=1;i<n+1;i++)
          scanf("%d",&t[i]);
        for(i=0;i<n;i++)
        {
          move=t[i+1]-t[i];
          if(move<0)
                move=move*-1;
          t_move+=move;
        }
        printf("Total head movement: %d\n",t_move);
        avg_move=(float)t_move/n;
        printf("Average head movement: %f\n",avg_move);
}
```

OUTPUT

```
midhun@midhun:~/Desktop/os$ ./a.out
Enter current head position: 55
Enter number of tracks: 6
Enter tracks to be traversed: 122 98 100 41 7 19
Total head movement: 198
Average head movement: 33.000000
```

SCAN Disk Scheduling

```
#include<stdio.h>
main()
{
  int t[20], d[20], h, i, j, n, temp, k, atr[20], tot, p, sum=0,end=199;
  printf("Enter the no of tracks to be traversed : ");
  scanf("%d" ,&n);
  printf("Enter the position of head : ");
  scanf("%d" ,&h);
  t[0]=0;t[1]=h;
  t[n+2]=end;
  printf("Enter the tracks : ");
  for(i=2;i<n+2;i++)
      scanf("%d" ,&t[i]);
  for(i=0;i<n+2;i++)
  {
      for( j=0; j<(n+2)-i-1; j++)
      {
            if(t[j]>t[j+1])
            {
              temp=t[j];
                         t[j]=t[j+1];
              t[j+1]=temp;
            }
      }
  }
  for(i=0;i<n+2; i++)
      if(t[i]==h)
      {
        k=i;
        p=0;
      }
  if(h<(end-h))
  {
      for(i=k;i>=0;i--)
      {
        atr[p]=t[i];
        p++;
      }
      for(i=k+1;i<n+2;i++)
      {
            atr[p]=t[i];
            p++;
      }
  }
  else
  {
      for (i=k;i<=n+2;i++)
            {
      atr[p]=t[i];
        p++;
      }
      for(i=k-1;i>0;i--)
      {
      atr[p]=t[i];
              p++;
     }
  }
  printf("Scheduling order : \n");
  for (p=0;p<n+2;p++)
          printf("%d \t",atr[p]);
  for (j=0; j<n+1; j++)
  {
          d[j]=0;
          if(atr[j]>atr[j+1])
            d[j]=atr[j]-atr[j+1];
          else d[j]=atr[j+1]-atr[j];
          sum+=d[j];
  }

  printf("\nTotal head movements:%d\n",sum);
 }
```

OUTPUT

```
midhun@midhun:~/Desktop/os$ ./a.out
Enter the no of tracks to be traversed : 8
Enter the position of head : 25
Enter the tracks : 10 50 40 60 20 70 80 30
Scheduling order :
25      20      10      0       30      40      50      60      70      80
Total head movements:105
```

CSCAN Disk scheduling

```
#include<stdio.h>
void main()
{
    int t[20],d[20],h,i,j,n,temp,k,atr[20],tot,p,sum=0,end=199;
    printf("\nEnter position of head: ");
    scanf("%d",&h);
    printf("Enter no of tracks to be traversed: ");
    scanf("%d",&n);
    t[0]=0;
    t[1]=h;
    t[n+2]=end;
    printf("\Enter tracks: ");
    for(i=2;i<n+2;i++)
             scanf("%d",&t[i]);
    for(i=0;i<n+2;i++)
    {
            for(j=0;j<(n+2)-i-1;j++)
            {
             if(t[j]>t[j+1])
             {
              temp=t[j];
              t[j]=t[j+1];
              t[j+1]=temp;
             }
            }
    }
    for(i=0;i<n+2;i++)
    {
      if(t[i]==h)
      {
            k=i;
            p=0;
      }
    }
    if(h<(end-h))
    {
      for(i=k;i>=0;i--)
      {
            atr[p]=t[i];
            p++;
      }
      for(i=n+2;i>k;i--)
      {
            atr[p]=t[i];
            p++;
```

```
 }
}
else
{
 for(i=k;i<=n+2;i++)
 {
     atr[p]=t[i];
     p++;
 }
 for(i=0;i<k;i++)
 {
     atr[p]=t[i];
     p++;
 }
}
printf("Scheduling order: \n");
for(p=0;p<=n+2;p++)
     printf("%d \t",atr[p]);
for(j=0;j<n+2;j++)
{
     d[j]=0;
     if(atr[j]>atr[j+1])
      d[j]=atr[j]-atr[j+1];
     else
      d[j]=atr[j+1]-atr[j];
sum+=d[j];
}
printf("\nTotal head movements: %d\n",sum);
}
```

OUTPUT

```
midhun@midhun:~/Desktop/os$ ./a.out

Enter position of head: 25
Enter no of tracks to be traversed: 8
Enter tracks: 50 40 60 20 70 80 30 10

Scheduling order:
25      20      10      0       199     80      70      60      50      40      30
Total head movements: 393
```

BANKERS ALGORITHM

```
#include<stdio.h>
struct pro
  {
        int allo[10],need[10],max[10],flag;
  };
void main()
{
   struct pro p[10];
   int avail[10],safe[10],i,j,pn,rn,pid,new,rcount,progress,dead,s,p_count;
   printf("Enter number of processes:\t");
   scanf("%d",&pn);
   printf("Enter number of resources:\t");
   scanf("%d",&rn);
   for(i=0;i<pn;i++)
   {
        printf("\nProcess %d details.\n",i);
        printf("Enter allocation:\t");
        for(j=0;j<rn;j++)
          scanf("%d",&p[i].allo[j]);
        printf("Enter max requirements:\t");
        for(j=0;j<rn;j++)
          scanf("%d",&p[i].max[j]);
        p[i].flag=0;
   }
   printf("\nEnter available:\t");
   for(j=0;j<rn;j++)
     scanf("%d",&avail[j]);
   printf("\nNew resource request: ");
   printf("\nEnter pid:\t");
   scanf("%d",&pid);
   printf("\nEnter resource request:\t");
   for(j=0;j<rn;j++)
   {
        scanf("%d",&new);
        p[pid].allo[j]+=new;
        avail[j]-=new;
   }
   for(i=0;i<pn;i++)
        for(j=0;j<rn;j++)
            p[i].need[j]=p[i].max[j]-p[i].allo[j];
```

```
s=0;p_count=0;
while(p_count<pn)    {
    progress=0;             for(i=0;i<pn;i++)
    {
      if(p[i].flag==0)
      {
        rcount=0;
        for(j=0;j<rn;j++)
        {
            if(p[i].need[j]<=avail[j])
                rcount++;
            else
                break;
        }
        if(rcount==rn)
        {
            p_count++;
            p[i].flag=1;
            printf("\nP%d is visited",i);
            safe[s]=i;
            s++;
            printf("\tAvailable mem: ( ");
            for(j=0;j<rn;j++)
            {
                avail[j]+=p[i].allo[j];
                printf("%2d ",avail[j]);
            }
            printf(")");
            progress=1;

        }
      }
    }
    if(progress==0)
    {   dead=1;break;}
}
if(dead==1)
    printf("\n DEADLOCK!");
else
{
    printf("\n\n System is in Safe state\n Safe Sequence: ");
    for(i=0;i<s;i++)
        printf("P%d ",safe[i]);
```

```
 }

 char * print[4]={"Name","Max","Allocation","Need"};
 printf("\n\n%-7s%-9s%-16s%-10s",print[0],print[1],print[2],print[3]);
 printf("\n");
 for(i=0;i<pn;i++)
 {
        printf("P%-6d",i);
        for(j=0;j<rn;j++)
                printf("%-2d",p[i].max[j]);
        printf("\t");
        for(j=0;j<rn;j++)
                printf("%-2d",p[i].allo[j]);
        printf("\t\t");
        for(j=0;j<rn;j++)
                printf("%-2d",p[i].need[j]);
        printf("\n");
 }
}
```

OUTPUT

```
midhun@midhun:~/Desktop/os$ ./bankers
Enter number of processes:        5
Enter number of resources:        3

Process 0 details.
Enter allocation:      0 1 0
Enter max requirements:   7 5 3

Process 1 details.
Enter allocation:      2 0 0
Enter max requirements:   3 2 2

Process 2 details.
Enter allocation:      3 0 2
Enter max requirements:   9 0 2

Process 3 details.
Enter allocation:      2 1 1
Enter max requirements:   2 2 2

Process 4 details.
```

```
Enter allocation:       0 0 2
Enter max requirements:    4 3 3

Enter available:        3 3 2

New resource request:
Enter pid:      1

Enter resource request:      1 0 2

P1 is visited   Available mem: (  5  3  2 )
P3 is visited   Available mem: (  7  4  3 )
P4 is visited   Available mem: (  7  4  5 )
P0 is visited   Available mem: (  7  5  5 )
P2 is visited   Available mem: ( 10  5  7 )

 System is in Safe state
 Safe Sequence: P1 P3 P4 P0 P2

Name   Max     Allocation      Need
P0     7 5 3   0 1 0           7 4 3
P1     3 2 2   3 0 2           0 2 0
P2     9 0 2   3 0 2           6 0 0
P3     2 2 2   2 1 1           0 1 1
P4     4 3 3   0 0 2           4 3 1
```