OS LAB MANUAL

Module - 1				
Week	Name of the Experiments	CLA	Assessments & marks distribution	CLA marks
1	Exploring Unix Commands	1	Objective & Procedure write up including outcomes - 4 Marks Experimentation and data collection - 4 Marks Computation of results - 4 Marks Analysis of results and interpretation - 4 Marks Viva voce - 4 Marks	20Marks for each CLA
2	Exploring Unix Commands	2		
3	Design & Development of program using Shell Scipt	3		
4	Evaluation of various process Scheduling Algorithms	4		
Module - 2				
1	Applying various Deadlock Prevention & Avoidance Algorithms	1	Objective & Procedure write up including outcomes - 4 Marks Experimentation and data collection - 4 Marks Computation of results - 4 Marks Analysis of results and interpretation - 4 Marks Viva voce - 4 Marks	20 Marks for each CLA
2	Implementation of Page Replacement Algorithm using FIFO, LRU	2		
3	Analyzing of various Memory management Techniques	3		
4	Implementation of Page Replacement Algorithm using OPTIMAL	4		
5	Implementation of Disk scheduling algorithm			
6	Revision	5		

			TOTAL MARKS	**160 Marks**
	CLA Marks Distribution	Marks	Faculties In-charge	Signatures
	Per CLA	20	Ms. G. Parmila	
	TOTAL Number of CLA's	8	Mr. Uttej Kumar .N	
	Total Marks	160	Mr. Vijay Babu P	
	T5 (TOTAL MARKS/8)	20	Mr. Badarsha	
			Mr. Subba Rao Maram	

<u>3)Design and Development of Progams using Shell Script:</u>

1. write Hello world program using shell script:

Ans:

```
echo -s &quot;Enter String: &quot; (-s is to read a String)
read name
echo &quot;$name&quot;
Output: Enter String: Hello world
Hello World
```

2. Add two numbes using Shell Script?

Ans:

```
echo -n &quot;Enter 1st number: &quot; (-n is to read a number)
read first_number
echo -n &quot;Enter 2nd number: &quot;
read second_number
sum=$(($first_number + $second_number))
echo &quot;Sum of $first_number and $second_number: &quot;$sum
output: Enter 1 st number: 10
Enter 2 nd number: 20
```

```
Sum of 10 and 20 : 30
```

3. Write a Program to swap two numbers Using Shell Script?

Ans:

```
#!/bin/bash
echo -n &quot;Enter number1:&quot;
read num1
echo -n &quot;Enter number2:&quot;
read num2
echo &quot;Before Swapping&quot;
echo &quot;Num1: $num1&quot;
echo &quot;Num2: $num2&quot;
num3=$num1
num1=$num2
num2=$num3
echo &quot;After Swapping&quot;
echo &quot;Num1: $num1&quot;
echo &quot;Num2: $num2&quot;
```

```
Output: Enter number1: 10
Enter number2: 20
Befor Swapping
Num1=10
Num2=20
After Swapping
Num1=20
Num2=10
```

4. Write Program to Find Armstrong number using Shell Script?

Ans:

```
#!/bin/bash

echo &quot;Enter a number: &quot;
read c
x=$c
sum=0
r=0
n=0
while [ $x -gt 0 ]
do
r=`expr $x % 10`
n=`expr $r \* $r \* $r`
sum=`expr $sum + $n`
x=`expr $x / 10`
done
if [ $sum -eq $c ]
then
echo &quot;It is an Armstrong Number.&quot;
else
echo &quot;It is not an Armstrong Number.&quot;
fi
```

Output:

```
Enter a number: 10
It is not an Armstrong Number.
Enter a number: 153
It is an Armstrong Number.
```

5. Fibonacci Series Program using Shell Script?

Ans:

```
echo -n &quot;Enter Number :&quot;
read N
echo -n &quot;Enter Num1 :&quot;
read a
echo -n &quot;Enter Num2 :&quot;
read b
echo &quot;The Fibonacci series is : &quot;
for (( i=0; i&lt;N; i++ ))
do
echo -n &quot;$a &quot;
fn=$((a + b))
a=$b
b=$fn
done
```

```
Output: Enter Number : 10
Enter Num1 : 1
Enter Num2 : 2
The Fibonacci series is : 1 2 3 5 8 13 21 34 55 89
```

6. Factorial Program using Shell Script?

Ans:

```
echo -n &quot;Enter a number :&quot;
read num

fact=1
while [ $num -gt 1 ]
do
fact=$((fact * num)) #fact = fact * num
num=$((num - 1)) #num = num - 1
done
echo “The Factorail of a number is : $fact”
```

```
Output: Enter a number : 4
The Factorial of a numbe is: 24
```

7. Palindrome Program using Shell Script?

Ans:

```
echo “Enter a Number: &quot;
read n
num=0
on=$n
while [ $n -gt 0 ]
do
num=$(expr $num \* 10)
k=$(expr $n % 10)
num=$(expr $num + $k)
```

```
n=$(expr $n / 10)
done
if [ $num -eq $on ]
then
echo palindrome
else
echo not palindrome
fi
Output: Enter a Number: 121
Palindrome
Enter a Number: 234
Not palindrome
```

4)Evaluation of various process scheduling algorithms:

1. Round robin algorithm using shell script:

```
echo Enter number of process:
read n
echo Enter quantum time:
read qt
echo Enter the burst time for each process:
for i in $(seq 1 1 $n)
do
echo -n Process $i : burst time:
read bt[i]
rbt[i]=${bt[i]}
done
p=$n
pt=0
while [[ $p>0 ]]
do
for i in $(seq 1 1 $n)
do
if [[ ${rbt[i]} -gt 0 ]]
then
if [[ ${rbt[i]} -le $qt ]]
then
pt=$((pt+rbt[i]))
rbt[i]=0
tat[i]=$pt
wt[i]=$((pt-bt[i]))
p=$((p-1))
else
rbt[i]=$((rbt[i]-qt))
pt=$((pt+qt))
fi
fi
done
done
for i in $(seq 1 1 $n)
```

```
do
echo process $i :waiting time ${wt[i]} turnaround time ${tat[i]}
done
```

Output:

```
vignan@vignan:~/os$ bash rr.sh
Enter number of process:
3
Enter quantum time:
2
Enter the burst time for each process:
Process 1 : burst time:10
Process 2 : burst time:6
Process 3 : burst time:3process 1 :waiting time 9 turnaround time 19
process 2 :waiting time 9 turnaround time 15
process 3 :waiting time 8 turnaround time 11
```

2. FCFS algorithm using shell script:

```
echo -n "Enter process number: "
read n1
BurstTime=()
WaitingTime=()
TurnAroundTime=()
for i in $(seq 1 1 $n1)
do
echo -n "Enter Burst Time for process:"
read bt
BurstTime+=($bt)
done
WaitingTime[0]=0
TurnAroundTime+=${BurstTime[0]}
TotalWaitingTime=0
TotalTurnAroundTime=${TurnAroundTime[0]}
for i in $(seq 1 1 $((n1-1)))
do
WaitingTime[$i]=$((${WaitingTime[$((i-1))]} + ${BurstTime[$((i-1))]}))
TurnAroundTime[$i]=$((${WaitingTime[$i]} + ${BurstTime[$i]}))
TotalWaitingTime=$(($TotalWaitingTime + ${WaitingTime[$i]}))
TotalTurnAroundTime=$(($TotalTurnAroundTime + ${TurnAroundTime[$i]}))
done
AvgWaitingTime=$(($TotalWaitingTime / $n1))
AvgTurnAroundTime=$(($TotalTurnAroundTime / $n1))
echo "PROCESS
BURSTTIME
WAITINGTIME
TURNAROUND TIME"
for(( i=0; i<$n1; i++))
do
echo "p:${i}
${BurstTime[$i]}
```

```
${WaitingTime[$i]}
$
{TurnAroundTime[$i]}"
done
echo Average Waiting Time : $AvgWaitingTime
echo Average Turn Around Time : $AvgTurnAroundTime
```

Output:

```
vignan@vignan:~/os$ bash fcfs.sh
Enter process number: 3
Enter Burst Time for process:9
Enter Burst Time for process:6
Enter Burst Time for process:8
PROCESS
BURSTTIME
WAITINGTIME
p:0
9
0
9
p:1
6
9
15
p:2
8
15
23
Average Waiting Time : 8
TURNAROUND TIMEAverage Turn Around Time : 15
```

3. SJF algorithm using shell script:

```
readarray fileDat < $1
quantum=${fileDat[${#fileDat[@]}-1]}
unset fileDat[${#fileDat[@]}-1]
processCount=${#fileDat[@]}
if [ $quantum -lt 3 ] || [ $quantum -gt 10 ] ; then
echo "ERROR: Quantum must be between 3 to 10"
exit
fi
function printProcess {
process=(${fileDat[$1]})
if [ -z $process ] ; then
return
fi
processName=${process[0]}
arrival=${process[1]}
burst=${process[2]}
priority=${process[3]}
echo Process Name: $processName
echo Arrival Time: $arrival
echo Burst Time: $burst
echo Priority: $priority
echo
}
count=0
let end=processCount-1
until [ $count -gt $end ]; do
printProcess $count
let count=count+1
done
echo Quantum: $quantum
echo
echo "~~~ Shortest Job First (SJF) Scheduling ~~~"
echo
sjfDat=("${fileDat[@]}")
shortestBurstIdx=0
currentTime=0
totalTurnaroundTime=0
waitingTime=0
echo "Grantt Chart: "
echo -n $currentTime' '
while [ ${#sjfDat[@]} -gt 0 ] ; do
shortestBurst=99999
count=0
until [ $count -gt $processCount ]; do
process=(${sjfDat[$count]})if [ -z $process ] ; then
let count=count+1
continue
fi
burst=${process[2]}
if [ $burst -lt $shortestBurst ]; then
shortestBurst=$burst
shortestBurstIdx=$count
fi
let count=count+1
done
chosenProcess=(${sjfDat[$shortestBurstIdx]})
processName=${chosenProcess[0]}
arrival=${process[1]}
burst=${chosenProcess[2]}
let currentTime=currentTime+burst
echo -n [$processName] $currentTime' '
let turnaroundTime=currentTime-arrival
let waitingTime=waitingTime+turnaroundTime-burst
let totalTurnaroundTime=totalTurnaroundTime+turnaroundTime
unset sjfDat[$shortestBurstIdx]
done
let avgWaitingTime=waitingTime/processCount
```

```
let avgTurnAroundTime=totalTurnaroundTime/processCount
echo "Total Turnaround Time :" $totalTurnaroundTime
echo "Average Turnaround Time :" $avgTurnAroundTime
echo "Total Waiting Time :" $waitingTime
echo "Average Waiting Time :" $avgWaitingTime
```

Output:

input.txt:

```
P1 2 6 7
P2 1 8 1
P3 18 4 2
P4 2 2 5
4
vignan@vignan:~/os$ ./sjf.sh input.txt
Process Name: P1
Arrival Time: 2
Burst Time: 6
Priority: 7
Process Name: P2
Arrival Time: 1
Burst Time: 8
Priority: 1
Process Name: P3
Arrival Time: 18
Burst Time: 4Priority: 2
Process Name: P4
Arrival Time: 2
Burst Time: 2
Priority: 5
Quantum: 4
~~~ Shortest Job First (SJF) Scheduling ~~~
```

Grantt Chart:

```
0 [P4] 2 [P3] 6 [P1] 12 [P2] 20 Total Turnaround Time : 40
Average Turnaround Time : 10
Total Waiting Time : 20
Average Waiting Time : 5
```

MODULE-2

1)Applying various Deadlock prevention and avoidance algorithms

1. Banker’s Algorithm(Prevention):

```
#include<stdio.h>
void main()
{
int allocated[15][15],max[15][15],need[15][15],avail[15],tres[15],work[15],flag[15];
```

```
int pno,rno,i,j,prc,count,t,total;
count=0;

printf("\n Enter number of process:");
scanf("%d",&pno);
printf("\n Enter number of resources:");
scanf("%d",&rno);
for(i=1;i<=pno;i++)
{
 flag[i]=0;
}
printf("\n Enter total numbers of each resources:");
for(i=1;i<= rno;i++)
 scanf("%d",&tres[i]);


printf("\n Enter Max resources for each process:");
for(i=1;i<= pno;i++)
{
 printf("\n for process %d:",i);
 for(j=1;j<= rno;j++)
  scanf("%d",&max[i][j]);
}


printf("\n Enter allocated resources for each process:");
for(i=1;i<= pno;i++)
{
 printf("\n for process %d:",i);
 for(j=1;j<= rno;j++)
  scanf("%d",&allocated[i][j]);


}

printf("\n available resources:\n");
for(j=1;j<= rno;j++)
{
 avail[j]=0;
 total=0;
 for(i=1;i<= pno;i++)
 {
  total+=allocated[i][j];
 }
 avail[j]=tres[j]-total;
 work[j]=avail[j];
 printf("    %d \t",work[j]);
}
do
```

```
{
for(i=1;i<= pno;i++)
{
 for(j=1;j<= rno;j++)
 {
  need[i][j]=max[i][j]-allocated[i][j];
 }
}
printf("\n Allocated matrix           Max           need");
for(i=1;i<= pno;i++)
{
 printf("\n");
 for(j=1;j<= rno;j++)
 {
  printf("%4d",allocated[i][j]);
 }
 printf("|");
 for(j=1;j<= rno;j++)
 {
  printf("%4d",max[i][j]);
 }
  printf("|");
 for(j=1;j<= rno;j++)
 {
  printf("%4d",need[i][j]);
 }
}
 prc=0;
 for(i=1;i<= pno;i++)
 {
  if(flag[i]==0)
  {
        prc=i;

        for(j=1;j<= rno;j++)
        {
        if(work[j]< need[i][j])
        {
        prc=0;
        break;
        }
        }
  }

  if(prc!=0)
  break;
 }
 if(prc!=0)
 {
   printf("\n Process %d completed",i);
   count++;
   printf("\n Available matrix:");
   for(j=1;j<= rno;j++)
   {
         work[j]+=allocated[prc][j];
         allocated[prc][j]=0;
         max[prc][j]=0;
         flag[prc]=1;
         printf("   %d",work[j]);
   }
  }
 }while(count!=pno&&prc!=0);

 if(count==pno)
  printf("\nThe system is in a safe state!!");
 else
  printf("\nThe system is in an unsafe state!!");
}
```

Output:

```
vignan@vignan:~/os$ ./a.out

 Enter number of process:5

 Enter number of resources:3

 Enter total numbers of each resources:10 5 7

 Enter Max resources for each process:
 for process 1:7 5 3

 for process 2:3 2 2

 for process 3:9 0 2

 for process 4:2 2 2

 for process 5:4 3 3

 Enter allocated resources for each process:
 for process 1:0 1 0

 for process 2:3 0 2

 for process 3:3 0 2

 for process 4:2 1 1

 for process 5:0 0 2
 available resources:
        2       3       0
 Allocated matrix        Max           need
   0   1   0|   7   5   3|   7   4   3
   3   0   2|   3   2   2|   0   2   0
   3   0   2|   9   0   2|   6   0   0
   2   1   1|   2   2   2|   0   1   1
   0   0   2|   4   3   3|   4   3   1
 Process 2 completed
 Available matrix:   5   3   2
 Allocated matrix        Max           need
   0   1   0|   7   5   3|   7   4   3
   0   0   0|   0   0   0|   0   0   0
   3   0   2|   9   0   2|   6   0   0
   2   1   1|   2   2   2|   0   1   1
   0   0   2|   4   3   3|   4   3   1
 Process 4 completed
 Available matrix:   7   4   3
 Allocated matrix        Max           need
   0   1   0|   7   5   3|   7   4   3
   0   0   0|   0   0   0|   0   0   0
   3   0   2|   9   0   2|   6   0   0
   0   0   0|   0   0   0|   0   0   0
   0   0   2|   4   3   3|   4   3   1
 Process 1 completed
 Available matrix:   7   5   3
 Allocated matrix        Max           need
   0   0   0|   0   0   0|   0   0   0
   0   0   0|   0   0   0|   0   0   0
   3   0   2|   9   0   2|   6   0   0
   0   0   0|   0   0   0|   0   0   0
   0   0   2|   4   3   3|   4   3   1
 Process 3 completed
 Available matrix:   10   5   5
 Allocated matrix        Max           need
   0   0   0|   0   0   0|   0   0   0
   0   0   0|   0   0   0|   0   0   0
   0   0   0|   0   0   0|   0   0   0
   0   0   0|   0   0   0|   0   0   0
   0   0   2|   4   3   3|   4   3   1
 Process 5 completed
 Available matrix:   10   5   7
```

2.Dining Philosophers(Avoidence):

```
#include<stdio.h>
#define n 4
int compltedPhilo = 0,i;
struct fork{
    int taken;
}ForkAvil[n];
struct philosp{
    int left;
    int right;
}Philostatus[n];
void goForDinner(int philID){
    if(Philostatus[philID].left==10 && Philostatus[philID].right==10)
    printf("Philosopher %d completed his dinner\n",philID+1);
    else if(Philostatus[philID].left==1 && Philostatus[philID].right==1){
      printf("Philosopher %d completed his dinner\n",philID+1);
      Philostatus[philID].left = Philostatus[philID].right = 10;
      int otherFork = philID-1;
      if(otherFork== -1)
        otherFork=(n-1);
      ForkAvil[philID].taken = ForkAvil[otherFork].taken = 0;
      printf("Philosopher %d released fork %d and fork %d\n",philID+1,philID+1,otherFork+1);
      compltedPhilo++;
    }
    else if(Philostatus[philID].left==1 && Philostatus[philID].right==0){
        if(philID==(n-1)){
          if(ForkAvil[philID].taken==0){
            ForkAvil[philID].taken = Philostatus[philID].right = 1;
            printf("Fork %d taken by philosopher %d\n",philID+1,philID+1);
          }else{
            printf("Philosopher %d is waiting for fork %d\n",philID+1,philID+1);
          }
        }else{
          int dupphilID = philID;
          philID-=1;

          if(philID== -1)
            philID=(n-1);

          if(ForkAvil[philID].taken == 0){
            ForkAvil[philID].taken = Philostatus[dupphilID].right = 1;
            printf("Fork %d taken by Philosopher %d\n",philID+1,dupphilID+1);
          }else{
            printf("Philosopher %d is waiting for Fork %d\n",dupphilID+1,philID+1);
          }
        }
      }
      else if(Philostatus[philID].left==0){
          if(philID==(n-1)){
            if(ForkAvil[philID-1].taken==0){
              ForkAvil[philID-1].taken = Philostatus[philID].left = 1;
              printf("Fork %d taken by philosopher %d\n",philID,philID+1);
            }else{
              printf("Philosopher %d is waiting for fork %d\n",philID+1,philID);
```

```
                        }
                    }else{
                        if(ForkAvil[philID].taken == 0){
                            ForkAvil[philID].taken = Philostatus[philID].left = 1;
                            printf("Fork %d taken by Philosopher %d\n",philID+1,philID+1);
                        }else{
                            printf("Philosopher %d is waiting for Fork %d\n",philID+1,philID+1);
                        }
                    }
        }else{}
}
int main(){
         for(i=0;i<n;i++)
        ForkAvil[i].taken=Philostatus[i].left=Philostatus[i].right=0;
         while(compltedPhilo<n){
                  for(i=0;i<n;i++)
            goForDinner(i);
                  printf("\nTill now num of philosophers completed dinner are %d\n\n",compltedPhilo);
         }

         return 0;
}
```

Output:

```
vignan@vignan:~$ ./a.out
Fork 1 taken by Philosopher 1
Fork 2 taken by Philosopher 2
Fork 3 taken by Philosopher 3
Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 0

Fork 4 taken by Philosopher 1
Philosopher 2 is waiting for Fork 1
Philosopher 3 is waiting for Fork 2
Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 0

Philosopher 1 completed his dinner
Philosopher 1 released fork 1 and fork 4
Fork 1 taken by Philosopher 2
Philosopher 3 is waiting for Fork 2
Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 1

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 2 released fork 2 and fork 1
Fork 2 taken by Philosopher 3
Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 2

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Philosopher 3 released fork 3 and fork 2
Fork 3 taken by philosopher 4

Till now num of philosophers completed dinner are 3

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Fork 4 taken by philosopher 4

Till now num of philosophers completed dinner are 3

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Philosopher 4 completed his dinner
Philosopher 4 released fork 4 and fork 3

Till now num of philosophers completed dinner are 4
```

2.Implementation of Page Replacement Algorithm using FIFO, LRU

1. Page Replacement Algorithm using FIFO:

```
 #include<stdio.h>
int main()
{
int i,j,n,a[50],frame[10],no,k,avail,count=0;
            printf("\n ENTER THE NUMBER OF PAGES:\n");
             scanf("%d",&n);
            printf("\n ENTER THE PAGE NUMBER :\n");
            for(i=1;i<=n;i++)
            scanf("%d",&a[i]);
            printf("\n ENTER THE NUMBER OF FRAMES :");
            scanf("%d",&no);
            for(i=0;i<no;i++)
            frame[i]= -1;
            j=0;
            printf("\tref string\t page frames\n");
            for(i=1;i<=n;i++)
           {
               printf("%d\t\t",a[i]);
               avail=0;
               for(k=0;k<no;k++)
                 if(frame[k]==a[i])
               avail=1;
               if (avail==0)
               {
                      frame[j]=a[i];
                        j=(j+1)%no;
                       count++;
                      for(k=0;k<no;k++)
                      printf("%d\t",frame[k]);
                 }
               printf("\n");
                 }
              printf("Page Fault Is %d",count);
              return 0;
}
```

Output:

```
vignan@vignan:~$ ./a.out

 ENTER THE NUMBER OF PAGES:
20

 ENTER THE PAGE NUMBER :
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

 ENTER THE NUMBER OF FRAMES :3
         ref string     page frames
7                7       -1      -1
0                7       0       -1
1                7       0       1
2                2       0       1
0
3                2       3       1
0                2       3       0
4                4       3       0
2                4       2       0
3                4       2       3
0                0       2       3
3
2
1                0       1       3
2                0       1       2
0
```

```
1
7          7    1    2
0          7    0    2
1          7    0    1
```

2. Page Replacement Algorithm using LRU:

```c
#include <stdio.h>
//user-defined function
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;
}
```

Output:

```
vignan@vignan:~$ ./a.out
Enter number of frames: 3
Enter number of pages: 10
Enter reference string: 7 5 9 4 3 7 9 6 2 1

7       -1      -1
7       5       -1
7       5       9
4       5       9
4       3       9
```

```
4	3	7
9	3	7
9	6	7
9	6	2
1	6	2
Total Page Faults = 10
```

3. Analyzing of various Memory management Techniques:

1. A program to simulate Paging technique of memory management.

```
#include<stdio.h>
main()
{
 int np,ps,i;
 int *sa;
 printf("enter how many pages\n");
 scanf("%d",&np);
 printf("enter the page size \n");
 scanf("%d",&ps);
 sa=(int*)malloc(2*np);
 for(i=0;i<np;i++)
 {
  sa[i]=(int)malloc(ps);
  printf("page%d\t address %u\n",i+1,sa[i]);
 }
}
```

Output:

```
vignan@vignan:~$ ./a.out
enter how many pages
3
enter the page size
4
page1	address 151244824
page2	address 151244840
page3	address 151244856
```

2. A program to simulate segmentation .

```
#include <stdio.h>
int main()
{
    int n,nm,p,x=0,y=1,t=300,of,i;
    printf("Enter the memory size:\n");
    scanf("%d",&nm);
    printf("Enter the no.of segments:\n");
    scanf("%d",&n);
    int s[n];
    for(i=0;i<n;i++)
    {
        printf("enter the segment size of %d:",i+1);
        scanf("%d",&s[i]);
        x+=s[i];
        if(x>nm)
        {
            printf("memory full segment %d is not allocated",i+1);
            x-=s[i];
            s[i]=0;
        }
    }
    printf("-----OPERATIONS------");
    while(y==1)
    {
        printf("enter the no.of operations:\n");
        scanf("%d",&p);
        printf("enter the offset:");
        scanf("%d",&of);
        if(s[p-1]==0)
        {
            printf("segment is not allocated\n");
        }
        else if(of>s[p-1])
        {
            printf("out of range!..");
        }
        else
        {
            printf("the segment %d the physical address is ranged from %d to %d\n the address of operation
is\n",p,t,t+s[p-1],t+of);
        }
        printf("press 1 to continue");
        scanf("%d",&y);
    }
}
```

Output:

```
vignan@vignan:~$ ./a.out
Enter the memory size:
10
Enter the no.of segments:
4
enter the segment size of 1:5
enter the segment size of 2:2
enter the segment size of 3:1
enter the segment size of 4:2
-----OPERATIONS------enter the no.of operations:
```

3

4 .Implementation of Page Replacement Algorithm using OPTIMAL

```
#include<stdio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
int recent[10],optcal[50],count=0;
int optvictim();
void main()
{
  printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHN");
  printf("\n................................");
  printf("\nEnter the no.of frames");
  scanf("%d",&nof);
  printf("Enter the no.of reference string");
  scanf("%d",&nor);
  printf("Enter the reference string");
  for(i=0;i<nor;i++)
     scanf("%d",&ref[i]);
  printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHM");
  printf("\n................................");
  printf("\nThe given string");
  printf("\n....................\n");
  for(i=0;i<nor;i++)
     printf("%4d",ref[i]);
  for(i=0;i<nof;i++)
  {
     frm[i]=-1;
     optcal[i]=0;
  }
  for(i=0;i<10;i++)
     recent[i]=0;
  printf("\n");
  for(i=0;i<nor;i++)
  {
    flag=0;
    printf("\n\tref no %d ->\t",ref[i]);
    for(j=0;j<nof;j++)
    {
      if(frm[j]==ref[i])
      {
        flag=1;
        break;
      }
    }
    if(flag==0)
    {
      count++;
      if(count<=nof)
         victim++;
      else
         victim=optvictim(i);
      pf++;
      frm[victim]=ref[i];
      for(j=0;j<nof;j++)
        printf("%4d",frm[j]);
    }
  }
  printf("\n Number of page faults: %d",pf);
}
int optvictim(int index)
{
  int i,j,temp,notfound;
  for(i=0;i<nof;i++)
  {
    notfound=1;
    for(j=index;j<nor;j++)
      if(frm[i]==ref[j])
      {
         notfound=0;
         optcal[i]=j;
         break;
      }
    if(notfound==1)
        return i;
  }
  temp=optcal[0];
  for(i=1;i<nof;i++)
     if(temp<optcal[i])
        temp=optcal[i];
  for(i=0;i<nof;i++)
     if(frm[temp]==frm[i])
        return i;
 return 0;
}
```

Output:

```
vignan@vignan:~$ ./a.out
 OPTIMAL PAGE REPLACEMENT ALGORITHN
................................
Enter the no.of frames3
Enter the no.of reference string6
Enter the reference string6 5 4 3 2 1

 OPTIMAL PAGE REPLACEMENT ALGORITHM
...............................
The given string
....................
   6   5   4   3   2   1
```

```
        ref no 6 ->    6  -1  -1
        ref no 5 ->    6   5  -1
        ref no 4 ->    6   5   4
        ref no 3 ->    3   5   4
        ref no 2 ->    2   5   4
        ref no 1 ->    1   5   4
 Number of page faults: 6
```

5. Implementation of Disk scheduling algorithm

1. FCFS Disk Scheduling Algorithm:

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

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

}
```

Output:

```
FCFS Disk Scheduling Algorithm:
vignan@vignan:~$ ./a.out
Enter the number of Requests
8
Enter the Requests sequence
95 180 34 119 11 123 62 64
Enter initial head position
50
Total head moment is 644
```

2. SSTF Disk Scheduling Algorithm:

```
#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);
    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];
        RQ[index]=1000;
        count++;
    }
    printf("Total head movement is %d",TotalHeadMoment);
    return 0;
}
```

Output:

```
vignan@vignan:~$ ./a.out
Enter the number of Requests
8
Enter the Requests sequence
95
180
34
119
11
123
62
64
Enter initial head position
50
Total head movement is 236
```