ANNAMALAI UNIVERSITY

(Accredited with Grade 'A' by NAAC)

FACULTY OF ENGINEERING AND TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LABORATORY MANUAL

FOR

DSCP408 – OPERATING SYSTEMS LAB

Name of the Programme	:	B.E. CSE (Data Science)
Semester	:	IV
Course Code	:	DSCP408
Course Name	:	OPERATING SYSTEMS LAB
Year	:	2021-2022
Lab In-charge	:	Dr.T.S.Subashini, Professor

Annamalai University

Department of Computer Science and Engineering

Faculty of Engineering and Technology

VISION

To provide a congenial ambience for individuals to develop and blossom as academically superior,socially conscious and nationally responsible citizens.

MISSION

- Impart high quality computer knowledge to the students through a dynamic scholastic environment wherein they learn to develop technical, communication and leadership skills to bloom as a versatile professional.
- Develop life-long learning ability that allows them to be adaptive and responsive to the changes in career, society, technology, and environment.
- Build student community with high ethical standards to undertake innovative research and developmentin thrust areas of national and international needs.
- Expose the students to the emerging technological advancements for meeting the demands of the industry.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO	PEO Statements
PEO1	To prepare the graduates with the potential to get employed in the right role and/or become entrepreneurs to contribute to the society.
PEO2	To provide the graduates with the requisite knowledge to pursue higher education and carry out research in the field of Computer Science.
PEO3	To equip the graduates with the skills required to stay motivated and adapt to the dynamically changing world so as to remain successful in their career.
PEO4	To train the graduates to communicate effectively, work collaboratively and exhibit high levels of professionalism and ethical responsibility.

08PC507 – OPERATING SYSTEMS LAB

PROGRAMME: B.E. (CSE) – Data Science **SEMESTER: V**
YEAR : Third Year **BATCH: A & B**

LIST OF EXPERIMENTS

CYCLE-1

1. Write a C program to implement the Job Scheduling techniques.
2. Write a C program to implement the Disk scheduling techniques.
3. Write a C program to implement the Memory management techniques.
4. Write a C program to implement the Page replacement techniques.
5. Write a C program to implement the Producer consumer problem.
6. Write a C program to implement the Banker's algorithm.

CYCLE-2

7. Write a shell script to perform the file operations using UNIX commands.
8. Write a shell script to perform the operations of basic UNIX utilities.
9. Write a shell script for arrange 'n' numbers using 'awk'.
10. Write a shell script to perform ${}^{n}C_{r}$ calculation using recursion.
11. Write a shell script to display the numbers between 1 and 9999 in words.
12. Write a shell script for Palindrome Checking.

Staff In-charge Head of the Department

COURSE OUTCOME STATEMENTS WITH CO-PO MAPPING TABLE

COURSE OBJECTIVES:

- To understand the basic concepts such as techniques, management of operating systems.
- To understand Operating System features and its difference from structured design.
- To use the UNIX as a modeling and communication utilities.
- To utilize the step of the process to produce better software.

COURSE OUTCOMES:

At the end of this course, the students will be able to

1. Develop C programs for Job scheduling techniques, Disk scheduling techniques,Memory management techniques and for synchronization problems.
2. Develop Shell script to practice Unix commands and utilities.
3. Demonstrate an ability to listen and answer the viva questions related to programmingskills needed for solving real-world problems in Computer Science and Engineering.

Mapping of Course Outcomes with Programme Outcomes												
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**
CO1	2	2	3	2	-	-	-	-	-	-	-	-
CO2	1	2	3	-	-	-	-	-	-	-	-	-
CO3	2	2	-	-	-	-	-	-	-	2	-	2

Rubrics for CO3

Rubric for CO3 in Laboratory Courses				
Rubric	**Distribution of 10 Marks for CIE/SEE Evaluation Out of 40/60 Marks**			
	Up To 2.5 Marks	**Up To 5 Marks**	**Up To 7.5 Marks**	**Up To 10 marks**
Demonstrate an ability to listen and answer the viva questions related to programming skills needed for solving real-world problems in Computer Science and Engineering.	Poor listening and communication skills. Failed to relate the programming skills needed for solving the problem.	Showed better communication skill by relating the problem with the programming skills acquired but the description showed serious errors.	Demonstrated good communication skills by relating the problem with the programming skills acquired with few errors.	Demonstrated excellent communication skills by relating the problem with the programming skills acquired and have been successful in tailoring the description.

INDEX

JOB SCHEDULING ALGORITHMS

Ex. No : 1

Date:

AIM:

To write a C program to implement FCFS, SJF, Priority and Round Robin job scheduling techniques.

CONCEPTS USED:

Terms Used:

- **Throughput:** Number of processes that are completed per time unit
- **Turnaround time:** The interval from the time of submission of a process to the time ofcompletion
- **Waiting time:** Waiting time is the sum of the periods spent waiting in the ready queue.
- **Response time:** It is the time from the submission of a request until the first response is produced. It is the time it takes to start responding, not the time it takes to output the response.

Scheduling Algorithms:

1. **First-Come First-Served Scheduling:** CPU is allocated to the process that requests the CPU first.
2. **Shortest-Job-First Scheduling:** When the CPU is available, it is assigned to the process that has the smallest CPU burst. If the CPU bursts of two processes are the same, FCFS scheduling is used to break the tie.
3. **Priority Scheduling:** A priority is associated with each process, and the CPU is allocated to the process with the highest priority. Equal-priority processes are scheduled in FCFS order.
4. **Round-Robin Scheduling:** A small unit of time, called a time quantum or time slice, is defined. A time quantum is generally from 10 to 100 milliseconds. The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocatingthe CPU to each process for a time interval of up to 1time quantum.

(A) FIRST-COME FIRST-SERVED SCHEDULING

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
struct process
{
char name[10];
int hr,min,sec,burst,wait,arrival,exit;
};
void main()
{
struct process p[20],temp;
void read_details_of_process(struct process[],int);
void print_details_of_process(struct
process[],int,int);int calculate_waiting_time(struct
process[],int);
int n,total;
clrscr();
printf("\nEnter the number of process: ");
scanf("%d",&n);
read_details_of_process(p,n);
total=calculate_waiting_time(p,n);
print_details_of_process(p,n,total);
getch();
}
void read_details_of_process(struct process p[],int n)
{
int i,j;
printf("\nEnter the details of %d processes:
",n);for(i=0;i<n;i++)
{
```

```
printf("\n\nProcess %d:",(i+1));
printf("\nEnter process name: ");
scanf("%s",&p[i].name);
printf("Enter arrival time: ");
printf("\n\tEnter Hour: ");
scanf("%d",&p[i].hr);
label1:
if(p[i].hr<=24)
{
printf("\tEnter Minute: ");
scanf("%d",&p[i].min);
label2:
if(p[i].min<=60)
{
printf("\tEnter Second: ");
scanf("%d",&p[i].sec);
label3:
if(p[i].sec<=60)
{
printf("Enter the burst time(in terms of seconds):
");scanf("%d",&p[i].burst);
}
else
{
printf("Enter seconds <= 60: ");
scanf("%d",&p[i].sec);
goto label3;
}
}
else
{
```

```
printf("Enter Minutes <= 60: ");
scanf("%d",&p[i].min);
goto label2;
}
}
else
{
printf("Enter hour <= 24: ");
scanf("%d",&p[i].hr);
goto label1;
}
p[i].arrival=p[i].sec+(p[i].min*60)+(p[i].hr*3600);
}
}
int calculate_waiting_time(struct process p[],int n)
{
struct process temp;
int i,j,total=0,t;
p[0].exit=p[0].arrival+p[0].burst;
for(i=0;i<n-1;i++)
{
for(j=i+1;j<n;j++)
{
if(p[i].arrival>p[j].arrival)
{
temp=p[i];
p[i]=p[j];
p[j]=temp;
}
}
}
```

```
for(i=0;i<n;i++)
{
if(i==0)
p[i].wait=0;
else if(p[i].arrival>p[i-1].exit)
{
p[i].wait=0;
p[i].exit=p[i].arrival+p[i].burst;
}
else if(p[i].arrival>p[i-1].arrival&&p[i].arrival<p[i-1].exit)
{
t=p[i].arrival-p[i-1].arrival;
p[i].wait=p[i-1].wait-t+p[i-1].burst;
p[i].exit=p[i].arrival+p[i].wait+p[i].burst;
}
else
{
p[i].wait=p[i-1].wait+p[i-1].burst;
p[i].exit=p[i].arrival+p[i].wait+p[i].burst;
}
total+=p[i].wait;
}
return total;
}
void print_details_of_process(struct process p[],int n,int total)
{
int i,j;
clrscr();
printf("\nProcess Name\tArrival Time\tBurst Time\tWaiting Time");
for(i=0;i<n;i++)
{
```

```
printf("\n%s\t\t%d:%d:%d\t\t%d\t\t%d",p[i].name,p[i].hr,p[i].min,p[i].sec,p[i].burst,p[i].wait);
}
printf("\nTotal Waiting Time: %d",total);
printf("\nAverage Waiting Time: %0.2f",(total/(n*1.0)));
}
```

SAMPLE INPUT AND OUTPUT:

```
Enter the number of process: 4
Enter the details of 4 processes:
Process 1:
Enter process name: p1
Enter arrival time:
Enter Hour: 4
Enter Minute: 10
Enter Second: 10
Enter the burst time (in terms of seconds): 60
Process 2:
Enter process name: p2
Enter arrival time:
Enter Hour: 4
Enter Minute: 10
Enter Second: 15
Enter the burst time (in terms of seconds): 95
Process 3:
Enter process name: p3
Enter arrival time:
Enter Hour: 4
Enter Minute: 10
Enter Second: 30
Enter the burst time(in terms of seconds): 50
Process 4:
```

Enter process name: p4

Enter arrival time:

Enter Hour: 5

Enter Minute: 12

Enter Second: 15

Enter the burst time (in terms of seconds): 80

Process Name Arrival Time Burst Time Waiting Time

p1 4:10:10 60 0

p2 4:10:15 95 55

p3 4:10:30 50 135

p4 5:12:15 80 0

Total Waiting Time: 190

Average Waiting Time: 47.50

(B) SHORTEST JOB FIRST SCHEDULING

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
struct process
{
char name[10];
int burst,wait;
};
void main()
{
void read_details_of_process(struct process[],int);
int calculate_waiting_time(struct process[],int);
void print_details_of_process(struct
process[],int,int);struct process p[20];
int total,n;
clrscr();
printf("\nEnter the number of process: ");
scanf("%d",&n);
read_details_of_process(p,n);
total=calculate_waiting_time(p,n);
print_details_of_process(p,n,total);
getch();
}
void read_details_of_process(struct process p[],int n)
{
int i,j;
printf("\nEnter the details of %d processes:
",n);for(i=0;i<n;i++)
{
```

```
printf("\n\nProcess %d:",(i+1));
printf("\nEnter process name: ");
scanf("%s",&p[i].name);
printf("Enter the burst time: ");
scanf("%d",&p[i].burst);
}}
int calculate_waiting_time(struct process p[],int n)
{
int i,j,t,total=0;
struct process temp;
for(i=0;i<n-1;i++)
{
for(j=i+1;j<n;j++)
{
if(p[i].burst>p[j].burst)
{
temp=p[i];
p[i]=p[j];
p[j]=temp;
}}}
for(i=0;i<n;i++)
{
if(i==0)
p[i].wait=0;else
p[i].wait=p[i-1].wait+p[i-1].burst;
total+=p[i].wait;
}
return total;
}
void print_details_of_process(struct process p[],int n,int total)
```

```
{
int i;
clrscr();
printf("\nProcess Name\tBurst Time\tWaiting Time");
for(i=0;i<n;i++)
{
printf("\n%s\t\t%d\t\t%d",p[i].name,p[i].burst,p[i].wait);
}
printf("\nTotal Waiting Time: %d",total);
printf("\nAverage Waiting Time: %0.2f",(total/(n*1.0)));
}
```

SAMPLE INPUT AND OUTPUT:

```
Enter the number of process: 4
Enter the details of 4 processes:
Process 1:
Enter process name: p1
Enter the burst time: 60
Process 2:
Enter process name: p2
Enter the burst time: 35
Process 3:
Enter process name: p3
Enter the burst time: 15
Process 4:
Enter process name: p4
Enter the burst time: 75
```

Process Name	Burst Time	Waiting Time
p3	15	0
p2	35	15
p1	60	50
p4	75	110

Total Waiting Time: 175

Average Waiting Time: 43.75

(C) PRIORITY SCHEDULING

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
struct process
{
char name[10]; int
burst,wait,pri;
};
void main()
{
void read_details_of_process(struct process[],int);
int calculate_waiting_time(struct process[],int);
void print_details_of_process(struct
process[],int,int);struct process p[20];
int total,n;
clrscr();
printf("\nEnter the number of process: ");
scanf("%d",&n);
read_details_of_process(p,n);
total=calculate_waiting_time(p,n);
print_details_of_process(p,n,total);
getch();
}
void read_details_of_process(struct process p[],int n)
{
int i,j;
printf("\nEnter the details of %d processes:
",n);for(i=0;i<n;i++)
{
```

```
printf("\n\nProcess %d:",(i+1));
printf("\nEnter process name: ");
scanf("%s",&p[i].name);
printf("Enter the burst time: ");
scanf("%d",&p[i].burst);
printf("Enter the priority: ");
scanf("%d",&p[i].pri);
}
}
int calculate_waiting_time(struct process p[],int n)
{
int i,j,t,total=0;
struct process temp;
for(i=0;i<n-1;i++)
{
for(j=i+1;j<n;j++)
{
if(p[i].pri>p[j].pri)
{
temp=p[i];
p[i]=p[j];
p[j]=temp;
}
}
}
for(i=0;i<n;i++)
{
if(i==0)
p[i].wait=0;else
p[i].wait=p[i-1].wait+p[i-1].burst;
```

```
total+=p[i].wait;
}
return total;
}
void print_details_of_process(struct process p[],int n,int total)
{
int i;
clrscr();
printf("\nProcess Name\tBurst Time\tPriority\tWaiting Time");
for(i=0;i<n;i++)
{
printf("\n%s\t\t%d\t\t%d\t\t%d",p[i].name,p[i].burst,p[i].pri,p[i].wait);
}
printf("\nTotal Waiting Time: %d",total);
printf("\nAverage Waiting Time: %0.2f",(total/(n*1.0)));
}
```

SAMPLE INPUT AND OUTPUT:

```
Enter the number of process: 4
Enter the details of 4 processes:
Process 1:
Enter process name: p1
Enter the burst time: 50
Enter the priority: 3
Process 2:
Enter process name: p2
Enter the burst time: 45
Enter the priority: 4
Process 3:
Enter process name: p3
Enter the burst time: 34
```

Enter the priority: 2

Process 4:

Enter process name: p4

Enter the burst time: 56

Enter the priority: 1

Process Name	Burst Time	Priority	Waiting Time
p4	56	1	0
p3	34	2	56
p1	50	3	90
p2	45	4	140

Total Waiting Time: 286

Average Waiting Time: 71.50

(D) ROUND ROBIN SCHEDULING

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
struct process
{
char name[10];
int burst,wait;
};
void main()
{
void read_details_of_process(struct process[],int);
int calculate_waiting_time(struct process[],int);
void print_details_of_process(struct
process[],int,int);struct process p[20];
int total,n;
clrscr();
printf("\nEnter the number of process: ");
scanf("%d",&n);
read_details_of_process(p,n);
total=calculate_waiting_time(p,n);
print_details_of_process(p,n,total);
getch();
}
void read_details_of_process(struct process p[],int n)
{
int i,j;
printf("\nEnter the details of %d processes:
",n);for(i=0;i<n;i++)
{
```

```
printf("\n\nProcess %d:",(i+1));
printf("\nEnter process name: ");
scanf("%s",&p[i].name);
printf("Enter the burst time: ");
scanf("%d",&p[i].burst);
}
}
int calculate_waiting_time(struct process p[],int n)
{
int i,j,k,t,total=0;
int ts,max=0,m1,w,round;
int mod[20][20];
int tbur[20];
struct process temp;
printf("\nEnter the time slice: ");
scanf("%d",&ts);
for(i=0;i<n;i++)
{
tbur[i]=p[i].burst;
if(max<tbur[i])
max=tbur[i];
}
round=max/ts+1;
for(i=1;i<=round;i++)
{
for(j=0;j<n;j++)
{
if(tbur[j]!=0)
{
m1=tbur[j]-ts;
if(m1>0)
```

```
{
mod[i][j]=ts;
tbur[j]=tbur[j]-ts;
}
else
{
mod[i][j]=tbur[j];tbur[j]=0;
}
}
else
mod[i][j]=0;
}
}
for(k=0;k<n;k++)
{
w=0;
p[k].wait=0;
for(i=1;i<=round;i++)
{
if(mod[i][k]==ts)
{
for(j=0;j<n;j++)
{
if(k!=j)
w=w+mod[i][j];
}
}
else if(mod[i][k]!=0)
{
for(j=0;j<k;j++)
```

```
w=w+mod[i][j];
}
}
p[k].wait=w;
total+=p[k].wait;
}
return total;
}
void print_details_of_process(struct process p[],int n,int total)
{
int i;
clrscr();
printf("\nProcess Name\tBurst Time\tWaiting Time");
for(i=0;i<n;i++)
{
printf("\n%s\t\t%d\t\t%d",p[i].name,p[i].burst,p[i].wait);
}
printf("\nTotal Waiting Time: %d",total);
printf("\nAverage Waiting Time: %0.2f",(total/(n*1.0)));
}
```

SAMPLE INPUT AND OUTPUT:

```
Enter the number of process: 3
Enter the details of 3 processes:
Process 1:
Enter process name: p1
Enter the burst time: 24
Process 2:
Enter process name: p2
Enter the burst time: 3
Process 3:
```

Enter process name: p3

Enter the burst time: 3

Enter the time slice: 4

Process Name	Burst Time	Waiting Time
p1	24	6
p2	3	4
p3	3	7

Total Waiting Time: 17

Average Waiting Time: 5.67

RESULT:

Thus the C program to implement the FCFS, SJF, Priority and Round Robin job schedulingtechnique is executed successfully and tested with various samples.

DISK SCHEDULING ALGORITHMS

Ex. No: 2

Date:

AIM:

To write C programs to implement FCFS, SSTF, SCAN and LOOK disk scheduling techniques.

CONCEPTS USED:

- **First Come First Served Scheduling:** First Request will be processed at first.
- **Shortest Seek Time First Scheduling:** The SSTF algorithm selects the request with the minimum seek time from the current head position. Since seek time increases with the number of cylinders traversed by the head, SSTF chooses the pending request closest tothe current head position.
- **Scan Scheduling:** In the SCAN algorithm, the disk arm starts at one end of the disk and move towards the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk.
- **Look Scheduling:** The arm goes only as far as the final request in each direction. Then, it reverses the direction immediately, without going all the way to the end of the disk.

A) FIRST COME FIRST SERVED SCHEDULING

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
int i,sum=0,n,st;
int a[20],b[20],dd[20];
clrscr();
do
{
printf("\nEnter the block number between 0 and 200: ");
```

```
scanf("%d",&st);
}while((st>=200)||(st<0));
printf("\nOur disk head is on the %d block",st);
a[0]=st;
printf("\nEnter the no. of request: ");
scanf("%d",&n);
printf("\nEnter request: ");
for(i=1;i<=n;i++)
{
printf("\nEnter %d request: ",i);
scanf("%d",&a[i]);
do
{
if((a[i]>200)||(a[i]<0))
{
printf("\nBlock number must be between 0 and 200!");
}}while((a[i]>200)||(a[i]<0));
}
for(i=0;i<=n;i++)
dd[i]=a[i];
printf("\n\t\tFIRST COME FIRST SERVE: ");
printf("\nDISK QUEUE:");
for(i=0;i<=n;i++)
printf("\t%d",a[i]);
printf("\n\nACCESS ORDER:");
for(i=0;i<=n;i++)
{
printf("\t%d",dd[i]);if(i!=n)
sum+=abs(dd[i]-dd[i+1]);
}
```

```
printf("\n\nTotal no. of head movements: %d",sum);
getch();
}
```

SAMPLE INPUT AND OUTPUT:

Enter the block number between 0 and 200:

53Our disk head is on the 53 block

Enter the no. of request: 8

Enter request:

Enter 1 request: 98

Enter 2 request: 183

Enter 3 request: 37

Enter 4 request: 122

Enter 5 request: 14

Enter 6 request: 124

Enter 7 request: 65

Enter 8 request: 67

FIRST COME FIRST SERVED:

DISK QUEUE:	53	98	183	37	122	14	124	65	67
ACCESS ORDER:	53	98	183	37	122	14	124	65	67

Total no. of head movements: 640

B) SHORTEST SEEK TIME FIRST SCHEDULING

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
int i,j,z,sum=0,c=0,n,n1,st,min;
int a[20],b[20],dd[20];
clrscr();
do
{
printf("\nEnter the block number between 0 and 200:
");scanf("%d",&st);
}while((st>=200)||(st<0));
printf("\nOur disk head is on the %d block",st);
a[0]=st;
printf("\nEnter the no. of request: ");
scanf("%d",&n);
printf("\nEnter request: ");
for(i=1;i<=n;i++)
{
printf("\nEnter %d request: ",i);
scanf("%d",&a[i]);
do
{
if((a[i]>200)||(a[i]<0))
{
printf("\nBlock number must be between 0 and 200!");
}}while((a[i]>200)||(a[i]<0));
}
```

```
for(i=0;i<=n;i++)
dd[i]=a[i];
n1=n;
b[0]=dd[0];
st=dd[0];
while(n1>0)
{
j=1;
min=abs(dd[0]-dd[1]);
for(i=2;i<n1+1;i++)
{
if(abs(st-dd[i])<=min)
{
min=abs(st-dd[i]);
j=i;
}}
c++;
b[c]=dd[j];
st=dd[j];
dd[0]=dd[j];
--n1;
for(z=j;z<n1+1;z++)
dd[z]=dd[z+1];
dd[z]='\0';
}
printf("\n\t\tSHORTEST SEEK TIME FIRST: ");
printf("\nDISK QUEUE:");
for(i=0;i<=n;i++)
printf("\t%d",a[i]);
printf("\n\nACCESS ORDER:");
for(i=0;i<=c;i++)
```

```
{
printf("\t%d",b[i]);
if(i!=c)
sum+=abs(b[i]-b[i+1]);
}
printf("\n\nTotal no. of head movements: %d",sum);
getch();
}
```

SAMPLE INPUT AND OUTPUT:

Enter the block number between 0 and 200:

53Our disk head is on the 53 block

Enter the no. of request: 8

Enter request:

Enter 1 request: 98

Enter 2 request: 183

Enter 3 request: 37

Enter 4 request: 122

Enter 5 request: 14

Enter 6 request: 124

Enter 7 request: 65

Enter 8 request: 67

SHORTEST SEEK TIME FIRST:

DISK QUEUE:	53	98	183	37	122	14	124	65	67
ACCESS ORDER:	53	65	67	37	14	98	122	124	183

Total no. of head movements: 236

C) SCAN SCHEDULING

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
int i,j,sum=0,c=0,n,st,temp,t;
int a[20],b[20],dd[20];
clrscr();
do
{
printf("\nEnter the block number between 0 and 200:
");scanf("%d",&st);
}while((st>=200)||(st<0));
printf("\nOur disk head is on the %d block",st);
a[0]=st;
printf("\nEnter the no. of request: ");
scanf("%d",&n);
printf("\nEnter request: ");
for(i=1;i<=n;i++)
{
printf("\nEnter %d request: ",i);
scanf("%d",&a[i]);
do
{
if((a[i]>200)||(a[i]<0))
{
printf("\nBlock number must be between 0 and 200!");
}
}while((a[i]>200)||(a[i]<0));
```

```
}
for(i=0;i<=n;i++)
dd[i]=a[i];
for(i=0;i<=n;i++)
for(j=i+1;j<=n;j++)
if(dd[i]>dd[j])
{
temp=dd[i];
dd[i]=dd[j];
dd[j]=temp;
}
for(i=0;i<=n;i++)
{
if(st==dd[i])
{
t=i+1;
b[c]=st;
for(j=i-1;j>=0;j--)
b[++c]=dd[j];
b[++c]=0;
for(j=t;j<=n;j++)
b[++c]=dd[j];
}
}
printf("\n\t\tSCAN TECHNIQUE: ");
printf("\nDISK QUEUE:");
for(i=0;i<=n;i++)
printf("\t%d",a[i]);
printf("\n\nACCESS ORDER:");
for(i=0;i<=c;i++)
{
```

```
printf("\t%d",b[i]);
if(i!=c)
sum+=abs(b[i]-b[i+1]);
}
printf("\n\nTotal no. of head movements: %d",sum);
getch();
}
```

SAMPLE INPUT AND OUTPUT:

Enter the block number between 0 and 200:

53Our disk head is on the 53 block

Enter the no. of request: 8

Enter request:

Enter 1 request: 98

Enter 2 request: 183

Enter 3 request: 37

Enter 4 request: 122

Enter 5 request: 14

Enter 6 request: 124

Enter 7 request: 65

Enter 8 request: 67

SCAN TECHNIQUE:

DISK QUEUE:	53	98	183	37	122	14	124	65	67	
ACCESS ORDER:	53	37	14	0	65	67	98	122	124	183

Total no. of head movements: 236

D) LOOK SCHEDULING

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
int i,j,sum=0,c=0,n,st,temp,t,s;
int  a[20],b[20],dd[20];
clrscr();
do
{
printf("\nEnter the block number between 0 and 200: ");
scanf("%d",&st);
}while((st>=200)||(st<0));
printf("\nOur disk head is on the %d block",st);
a[0]=st;
printf("\nEnter the no. of request: ");
scanf("%d",&n);
printf("\nEnter request: ");
for(i=1;i<=n;i++)
{
printf("\nEnter %d request: ",i);
scanf("%d",&a[i]);
do{
if((a[i]>200)||(a[i]<0))
{
printf("\nBlock number must be between 0 and 200!");
}}while((a[i]>200)||(a[i]<0));
}
i=0;i<=n;i++)
```

```
dd[i]=a[i];
s=a[0];
for(i=0;i<=n;i++)
for(j=i+1;j<=n;j++)
if(dd[i]>dd[j])
{
temp=dd[i];
dd[i]=dd[j];
dd[j]=temp;
}
for(i=0;i<=n;i++)
{
if(s==dd[i])
{
b[c]=st;
for(j=i-1;j>=0;j--)
b[++c]=dd[j];
b[++c]=200;
for(j=n;j>i;j--)
b[++c]=dd[j];
}}
printf("\n\t\tLOOK TECHNIQUE: ");
printf("\nDISK QUEUE:");
for(i=0;i<=n;i++)
printf("\t%d",a[i]);
printf("\n\nACCESS ORDER:");
for(i=0;i<=c;i++)
{
printf("\t%d",b[i]);
if(i!=c)
sum+=abs(b[i]-b[i+1]);
```

```
}
printf("\n\nTotal no. of head movements: %d",sum);
getch();
}
```

SAMPLE INPUT AND OUTPUT:

Enter the block number between 0 and 200:

53Our disk head is on the 53 block

Enter the no. of request: 8

Enter request:

Enter 1 request: 98

Enter 2 request: 183

Enter 3 request: 37

Enter 4 request: 122

Enter 5 request: 14

Enter 6 request: 124

Enter 7 request: 65

Enter 8 request: 67

LOOK TECHNIQUE:

DISK QUEUE:	53	98	183	37	122	14	124	65	67	
ACCESS ORDER:	53	37	14	200	183	124	122	98	67	65

Total no. of head movements: 360

RESULT:

Thus C programs to implement different disk scheduling techniques are written successfully andtested with various samples.

MEMORY MANAGEMENT TECHNIQUES

Ex. No: 3

Date:

Aim:

To simple the memory management schemes for the following

1. Paging
2. Demand paging
3. Segmentation

Algorithm:

Step 1: Start the program.

Step 2: Display the menu.

Step 3: Input choice for menu.

Step 4: If choice = 1 then passé ().

Step 5: If choice = 2 then demand (). **Step 6:** If choice = 3 then segment ().**Step 7:** Else go to step 8

Step 8: Stop the program

Page ()

Step 1: Start.

Step 2: Display the menu.

Step 3: If choice = 1 then start ().

Step 4: If choice = 2 then retrieve ().**Step 5:** Stop.

Store ()

Step 1: Start.

Step 2: Input total logical size should be power (step 2)

Step 3: Input passé size (should be power of 3).

Step 4: Total passé = Process/size (passé size).

Step 5: F = Process size % passé size.

Step 6: If F! = 0 Total passé = total passé+1.

Step 7: Input passé no corresponding for ms no.

Step 8: Print the physical address for corresponding from no

Step 9: Stop.

Retrieve ()

Step 1: Start.

Step 2: Input logical address to retrieval.

Step 3: Pass no = logical address/ passé size.

Step 4: Frame no = frame (pass no).

Step 5: Display comment = logical address % pass size

Step 6: The context in printed.

Step 7: Stop.

Segmentation ()

Step 1: Start.

Step 2: Input choice from menu.

Step 3: If choice = 1, Then store ().

Step 4: If choice = 2, Then Retrieval

().**Step 5:** Stop.

Store ()

Step 1: Start.

Step 2: Input logical size (No of segment).

Step 3: Input segmentation name.

Step 4: Input limit base array.

Step 5: Print physical memory arrays.

Step 6: Stop.

Demand ()

Step 1: Start.

Step 2: Display the menu.

Step 3: Input choice from the menu.

Step 4: If choice = 1 then store ().

Step 5: If choice = 2 then

retrieval().**Step 6:** Stop.

Store ()

Step 1: Start.

Step 2: Input logical memory size.

Step 3: Input contents.

Step 4: Input frame No, to store the Backup.

Step 5: Input choice of store the content in page table.

Step 6: If choice is % is getting the frame No.

Step 7: Else continue step 5 until the last Number is read.

Step 8: Print the physical memory arrays.

Step 9: Stop.

Retrieval ()

Step 1: Start.

Step 2: Input page number to Demand.

Step 3: If demand page is valid print array available and go to step 3.

Step 4: Stop.

Retrieval ()

Step 1: Start.

Step 2: Input segment No.

Step 3: Check whether the given segment displacement should be less than the limit.

Step 4: If Result is Error, print Address.

Step 5: Else add the displacement with

base.**Step 6:** Print the physical Address.

Step 7: Stop.

//Memory Management Techniques

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

```
{
struct page
{ char con[10];
}
page[10];
}
p[10];
struct seg
{ char name[10];
int limit,base;
}
seg[10];
int pf[10],con[20][20],val[20][20],stat[20];
int e=0,totpages,pasize,ps,pm,lm,totsegs,prs;
void main()
{
void paging(),segment(),dempaging();
int ch;
clrscr();
do
{
printf("\tmemory management menu");
printf("\n1.pageing\n2.segmentation\n3.demand pageing\n4.exit\n");
printf("\n Enter your choice:");
scanf("%d",&ch);
switch(ch)
{ case 1:paging();break;
case 2:segment();break;
case 3:dempaging();break;
case 4:exit(0);
}}
while(ch!=4);
}
void paging()
```

```
{
void pagestore(),pagerestore();
int ch;
do
{
printf("\n1.pagstore\n2.pagrestore\n3.return to main\nEnter your choice:");
scanf("%d",&ch);
switch(ch)
{  case  1:pagestore();break;
case   2:pagerestore();break;
default:continue;
}}
while(ch!=3);
}
void segment()
{
void segstore(),segrestore();
int ch;
do
{
printf("\n1.segment store\n2.segment restore\n3.return to
main");printf("\n Enter your choice:");
scanf("%d",&ch);
switch(ch)
{  case  1:segstore();break;
case   2:segrestore();break;
default:continue;
}}
while(ch!=3);
}
void dempaging()
```

```
{
void demstore(),demrestore();
int ch;
do
{
printf("\n1.demand paging-store\n2.demand paging-restore\n3.return to main\n");
printf("\nEnter your choice:");
scanf("%d",&ch);
switch(ch)
{ case 1:demstore();
break;
case 2:demrestore();
break;
default:continue;
}}
while(ch!=3);
}
void pagestore()
{int lms,i,j;
do
{
printf("\nEnter logical memory size:");
scanf("%d",&lms);
}while(lms%2!=0);
printf("\nenter process size:");
scanf("%d",&ps);
do
{
printf("\n Enter page size:");
scanf("%d",&pasize);
}
```

```
while(pasize%2!=0);
totpages=ps/pasize;
if(ps%pasize!=0)
totpages++;
for(i=0;i<totpages;i++)
{
printf("\n enter content of page[%d]:",i);
for(j=0;j<pasize;j++)
{
printf("\n Enter content[%d]:",j);
scanf("%s",p[i].page[j].con);
e++;
if(e==ps)
break;
}}
printf("\nEnter the frame numbers corresponding to the page numbers:\n");
for(i=0;i<totpages;i++)
{
b1:
printf("\n Enter the frame number of the page
%d:",i);scanf("%d",&pf[i]);
for(j=0;j<i;j++)
if(pf[j]==pf[i]) goto b1;
}
printf("\n Physical memory:\n");
for(i=0;i<totpages;i++)
{
printf("\n content of page[%d]:",i);
for(j=0;j<pasize;j++)
{
pm=pf[i]*pasize+j;
```

```
printf("\n frame no:%d",pf[i]);
printf("\n content[%d]:%s",pm,p[i].page[j].con);
e++;
if(e==ps)
break;
}
}}
void pagerestore()
{i
nt page,pn,po,fn;
printf("Enter the logical memory address:");
scanf("%d",&lm);
pn=lm/pasize;
po=lm%pasize;
fn=pf[pn];
pm=fn*pasize+po;
printf("\n logical memory address:%d",lm);
printf("\n page no:%d",pn);
printf("\n frame no: %d",fn);
printf("\n physical memory address: %d",pm);
printf("\n content:%s",p[pn].page[po].con);
}
void segstore()
{ char check(int);
char r;
int i,j;
printf(" Enter total no.of segment:");
scanf("%d",&totsegs);
for(i=0;i<totsegs;i++)
{
printf("\n Enter segment name:");
```

```
scanf("%s",seg[i].name);
b2:
printf("\n enter the limit of the segment:");
scanf("%d",&seg[i].limit);
printf("\n Enter the base of the segment:");
scanf("%d",&seg[i].base);
r=check(i);
if((r=='n'))
goto b2;
}
printf("\n physical memory\nsegment name\tlimit\t\tbase\n");
for(i=0;i<totsegs;i++)
{
printf("\n %d %s",i,seg[i].name);
printf("\t%d\t\t%d\n",seg[i].limit,seg[i].base);
}}
void segrestore()
{i
nt sn,so;
printf("\n Enter the segment no.to restore:");
scanf("%d",&sn);
if(sn>totsegs)
printf("\nsegment number does not exist");
b3:
printf("\n Enter the offset value:");
scanf("%d",&so);
if(so>seg[sn].limit)
goto b3;
printf("\n segment details\n");
printf("\n segment name:%s\n",seg[sn].name);
printf("\n segment base:%d\n",seg[sn].base);
```

```
printf("\n segment limit:%d\n",seg[sn].limit);
printf("\n equivalent physical address:%d",seg[sn].base+so);
} char check(int i)
{i
nt t,t1,j;
for(j=0;j<i;j++)
{t
=seg[j].base+seg[j].limit;
t1=seg[j+1].base+seg[j+1].limit;
if((seg[j].base==seg[j+1].base)||(seg[j+1].base<t))return 'b';
else if((t1<seg[j].base)||(t1<t))return 'y';
else return 'n';
} return 0;
}
void demstore()
{
int lms,i,j,k;
printf("\n Enter the total number of pages:");
scanf("%d",&prs);
for(i=0;i<prs;i++)
{
printf("\n Enter conment of page[%d]:",i+1);
scanf("%s",con[i]);
strcpy(val[i],con[i]);
}
printf("\n enter the frame no.of the pages \n type-1 for invalid pages");
for(i=0;i<prs;i++)
{
k1:
scanf("%d",&k);
for(j=0;j<i;j++)
```

```
{i f(k==pf[i])
{
printf("allocated frame");
goto k1;
}} pf[i]=k;
if(pf[i]==-1)stat[i]=0;
else stat[i]=1;
}
printf("\n physical memory\npage no\tcontent\tframeno\tstatus\n");
for(i=0;i<prs;i++)
{i
f(stat[i]==1)
printf("\n%d\t%s\t%d\tvalid\n",i,con[i],pf[i]);
else
printf("\n%d\t%s\tnull\tinvalid\n",i,con[i]);
}}
void demrestore()
{i
nt pno,i;
printf("\n Enter page no to retrive:");
scanf("%d",&pno);
if(stat[pno]==1)
{
printf("page%d is already losded \n",pno);
getch();
} else
{
printf("\n page fault occur:");
for(i=0;i<prs;i++)
{i
```

```
f(stat[pno]==-1)
{
stat[pno]=1;
pf[i]=pno;
break;
}}}
printf("\n physical memory\n page no\t content\t frame no\t status\n");
for(i=0;i<prs;i++)
{
printf("\n%d\t\t%s\t\t%d\t",i,con[i],pf[i]);
if(stat[i]==1)
printf("\tvalid");
else
printf("\tinvalid");
}}
```

Sample input and output

```
Memory management menu
  1. paging
  2. segmentation
  3. demand
  paging4.exit
Enter your choice:1
1.pagestore
2.pagerestore
3.return to main
Enter your choice:1
Enter logical memory size:20
enter process size:6
Enter page size:2
enter content of page[0]:
```

Enter content[0]:hello!

Enter content[1]:my

enter content of page[1]:

Enter content[0]:dear

Enter content[1]:friends

enter content of page[2]:

Enter content[0]:please

Enter content[1]:welcome

Enter the frame numbers corresponding to the page numbers:

Enter the frame number of the page 0:10

Enter the frame number of the page 1:20

Enter the frame number of the page 2:30

Physical memory:

content of page[0]:

frame no:10

content[20]:hello!

frame no:10

content[21]:my

content of page[1]:

frame no:20

content[40]:dear

frame no:20

content[41]:friends

content of page[2]:

frame no:30

content[60]:please

frame no:30

content[61]:welcome

1.pagestore

2.pagerestore 3.return

to main

Enter your choice:2
Enter the logical memory address:1
logical memory address:1
page no:0
frame no: 10
physical memory address: 21
content:my
1.pagestore
2.pagerestore
3.return to main
Enter your
choice:3
memory management menu
1.paging
2.segmentation
3.demand
paging4.exit
Enter your choice:2
1.segment store
2.segment restore
3.return to main
Enter your choice:1
Enter total no.of segment:4
Enter segment name:white
enter the limit of the segment:100
Enter the base of the segment:2000
Enter segment name:blue
enter the limit of the segment:200
Enter the base of the segment:1800
Enter segment name:green
enter the limit of the segment:300
Enter the base of the segment:1500

Enter segment name:red

enter the limit of the segment:400

Enter the base of the segment:1100

physical memory

segment	name	limit	base
0	white	100	2000
1	blue	200	1800
2	green	300	1500
3	red	400	1100

1.segment store

2. segment restore

3.return to main

Enter your

choice:2

Enter the segment no.to restore:2

Enter the offset value:100

segment details

segment name:green

segment base:1500

segment limit:300

equivalent physical address:1600

1.segment store

2.segment restore

3.return to main

Enter your choice:3

memory management menu

1.paging

2.segmentation

3.demand paging

4.exit

Enter your choice:3

1.demand paging-store

2.demand paging-restore

3.return to main

Enter your choice:1

Enter the total number of pages:4

Enter conment of page[1]:MCA

Enter conment of page[2]:ME

Enter conment of page[3]:BCA

Enter conment of page[4]:BE

enter the frame no.of the pages

type-1 for invalid pages

5

10

15

-1

physical memory

page no	content	frameno	status
0	MCA	5	valid
1	ME	10	valid
2	BCA	15	valid
3	BE	null	invalid

1.demand paging-store

2.demand paging-

restore3.return to main

Enter your choice:2

Enter page no to retrive:3

page fault occur:

physical memory

page no	content	frame no	status
0	MCA	5	valid
1	ME	10	valid
2	BCA	15	valid

3 BE -1

invalid1.demand paging-store

2.demand paging-

restore3.return to main

Enter your choice:2

Enter page no to retrive:1

page1 is already loaded

physical memory

page no	content	frame no	status
0	MCA	5	valid
1	ME	10	valid
2	BCA	15	valid
3	BE	-1	invalid

1.demand paging-store

2.demand paging-restore

3. return to main

Enter your choice:3

Memory management menu

1. paging

2. segmentation

3. demand

paging4.exit

Enter your choice:4

RESULT:

Thus the memory management techniques have been implemented and the output was verified.

PAGE REPLACEMENT TECHNIQUES

Ex.No:4

DATE:

AIM:

To write a c program to implement FIFO and LRU page replacement algorithm.

A) FIRST IN FIRST

OUTALGORITHM:

1. Start the process
2. Declare the size with respect to page length
3. Check the need of replacement from the page to memory
4. Check the need of replacement from old page to new page in memory
5. Form a queue to hold all pages
6. Insert the page require memory into the queue
7. Check for bad replacement and page fault
8. Get the number of processes to be inserted
9. Display the values
10. Stop the process

PROGRAM:

```
#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;
}
```

SAMPLE INPUT AND OUTPUT:

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

Page Fault is: 15

B) **LEAST RECENTLY USED**

ALGORITHM :

1. Start the process
2. Declare the size
3. Get the number of pages to be inserted
4. Get the value
5. Declare counter and stack
6. Select the least recently used page by counter value
7. Stack them according the selection.
8. Display the values
9. Stop the process

PROGRAM:

```
#include<stdio.h>
main()
{
int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];
printf("Enter no of pages:");
scanf("%d",&n);
printf("Enter the reference string:");
for(i=0;i<n;i++)  scanf("%d",&p[i]);
printf("Enter no of frames:");
scanf("%d",&f);
q[k]=p[k];
printf("\n\t%d\n",q[k]);
c++;
k++;
for(i=1;i<n;i++)
{
c1=0;
for(j=0;j<f;j++)
```

```
{
if(p[i]!=q[j])
c1++;
}
if(c1==f)
{
c++;
if(k<f)
{
q[k]=p[i];
k++;
for(j=0;j<k;j++)
printf("\t%d",q[j]);
printf("\n");
}
else
{
for(r=0;r<f;r++)
{
c2[r]=0;
for(j=i-1;j<n;j--)
{
if(q[r]!=p[j])
c2[r]++;
else
break;
}
}
for(r=0;r<f;r++)
b[r]=c2[r];
for(r=0;r<f;r++)
```

```
{
for(j=r;j<f;j++)
{
if(b[r]<b[j])
{
t=b[r];
b[r]=b[j];
b[j]=t;
}
}
}
for(r=0;r<f;r++)
{
if(c2[r]==b[0])
q[r]=p[i];
printf("\t%d",q[r]);
}
printf("\n");
}
}
}
printf("\nThe no of page faults is %d",c); }
```

OUTPUT:

```
Enter no of pages: 10
Enter the reference string: 7 5 9 4 3 7 9 6 2 1
Enter no of frames: 3
7
   7       5
   7       5       9
   4       5       9
   4       3       9
```

4	3	7
9	3	7
9	6	7
9	6	2
1	6	2

The no of page faults is 10

RESULT:

Thus FIFO and LRU page replacement techniques are well executed and verified.

IMPLEMENTATION OF PRODUCER CONSUMER PROBLEM

Ex.No:5

Date:

AIM:

To write a C program to implement producer consumer problem.

ALGORITHM:

Step 1: start

Step 2: display the menu and read

Step 3: If choice=1 then do the following steps

a) Get the process to be produced.

b) Check whether the process already

exists.If yes display the message

Else

Produce the process and display the process

list.Step 4: If choice=2 then do the following

steps

a) Get the process to be consumed

b) Check whether the process is already

producedIf yes consume the process

Else

Display the waiting liststep

5: stop.

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
struct prod
{
int s;
char wait[20][20];
};
static struct prod se={0};
char produce[20][20],consume[20];
int flag,i,j,z=1;
void main()
{
int ch;
void producer();
void consumer();
clrscr();
do
{
printf("\n\t\t MENU");
printf("\n\t\t ****");
printf("\n 1.producer");
printf("\n 2.consumer");
printf("\n 3.exit");
printf("\n enter your choice:");
scanf("%d",&ch);
switch(ch)
{
case 1:
```

```
producer();
break;
case 2:
consumer();
break;
case 3:
exit(0);
break;
}
}
while(ch!=3);
}
void producer()
{
flag=0;
printf("\n enter the producer process
name:");scanf("%s",&produce[++se.s]);
for(i=0;i<se.e;i++)
{
if(strcmp(produce[i],produce[se.e])==0)
{
printf("\n process already exist");
getch();
flag=1;
se.e--;
break;
}
}
for(i=0;i<se.e;i++)
{
if(strcmp(se.wait[i],produce[se.e])==0)
```

```
{
j=1;
printf("\n process %s now
consumed",produce[se.e]);se.e--;
flag=2;
break;
}
}
if(flag==1)
return;
else if(flag==2)
{
for(i=j;i<z;i++)
strcpy(se.wait[i],se.wait[i+1]);
z--;
}
else if(flag==0)
{
printf("list of produced process\n");
for(i=1;i<se.e;i++)
printf("%s\n",produce[i]);
}
}
void consumer()
{
flag=0;
printf("\n enter the consumer process name:);
scanf("%s",&consume);
for(i=1;i<se.e;i++)
{
if(strcmp(produce[i],consume)==0)
```

```
{
printf("\n process %s now
consumed",produce[i]);j=1;
flag=1;
break;
}
}
for(i=0;i<z;i++)
{
if(strcmp(produce[i],consume)==0)
{
printf("\n process already exists");
flag=2;
break;
}
}
if(flag==1)
{
for(i=1;i<se.e;i++)
strcpy(produce[i],produce[i+1]);
se.e--;
}
else if(flag==0)
{
strcpy(se.wait[++z],consume);
z++;
printf("list of waiting process\n");
for(i=1;i<z;i++)
printf("%s\n",se.wait[i]);
}
}
```

SAMPLE INPUT AND OUTPUT:

MENU

1. Producer
2. consumer
3. exit

enter your choice:1

enter the producer process name:p1

list of producer process:p1

MENU

1. producer
2. consumer
3. exit

enter your choice:1

enter the producer process name:p2

list of producer process:p1

p2

MENU

1. Producer
2. consumer
3. exit

enter your choice:1

enter the producer process name:p1

process already exists

MENU

1. Producer
2. consumer
3. exit

enter your choice:1

enter the producer process name:p3

list of producer process:p1

p2

p3

MENU

1. Producer
2. consumer
3. exit

enter your choice:2

enter the producer process name:p1

process p1 is consumed

MENU

1. Produucer
2. consumer
3. exit

enter your choice:2

enter the producer process name:p4

list of waiting process name:p4

MENU

1. producer
2. consumer
3. exit

enter your choice:3

RESULT

Thus the above producer consumer problem is well executed and verified.

IMPLEMENTATION OF BANKERS ALGORITHM

Ex.No:6

A) DEADLOCK DETECTION

Date:

To write a C program to implement deadlock detection.

CONCEPTS USED:

Terms Used:

- Available - A vector of length m indicates the number of available resources of each type.
- Allocation - An n x m matrix defines the number of resources of each type currentlyallocated to each process.
- Request - An n x m matrix indicates the current request of each process. If Request[i][j]equals k, then process Pi is requesting k more instances of resource type Rj.

Algorithm:

i. Let Work and Finish be vectors of length m and n, respectively. Initialize Work == Available. For i = 0 , 1 , . . . , n-1, if Allocation!= 0, then Finish[i] = false; otherwise, Finish[i] = true.

ii. Find an index i such that both

a. Finish[i] = false

b. Requesti <= Work

c. If no such exists, go to step 4.

iii. Work = Work + Allocationi

a. Finish[i] = true

b. Go to step 2.

iv. If Finish[i] == false, for some, 0=<i<n, then the system is in a deadlocked state.Moreover, if Finish[i] == false, then process Pi is deadlocked.

PROGRAM:

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

```
struct process
{
char pname[15];
int allocation[10],request[10],finish;
};
int available[10],avail[10];
int no_p,no_r,i,j,x,k;
struct process p[10],temp,*temp1;
char c;
char p_req[10];int
detect();
void resource_status(int[]);
void process_status();
void main()
{
clrscr();
printf("\nEnter number of processes: ");
scanf("%d",&no_p);
for(i=0;i<no_p;i++)
{
printf("\nEnter %d process name: ",(i+1));
scanf("%s",&p[i].pname);
}
printf("\nEnter number of resources: ");
scanf("%d",&no_r);
printf("\nEnter availability of each resource.");
for(i=0;i<no_r;i++)
{
printf("\n\tEnter availability of %d resource:
",(i+1));scanf("%d",&available[i]);
avail[i]=available[i];
```

```
}
for(i=0;i<no_p;i++)
{
printf("\n\nEnter allocated resource of process %d: ",(i+1));
for(j=0;j<no_r;j++)
{
lab2:
printf("\n\tResource %d---- >",(j+1));
scanf("%d",&p[i].allocation[j]);
if(p[i].allocation[j]>avail[j])
{
printf("\n\tAllocated resource is greater than available resource.Please enter smaller amount.");
goto lab2;
}
avail[j]=avail[j]-p[i].allocation[j];
}
}
printf("\n\nEnter request of each process: ");
for(i=0;i<no_p;i++)
{
printf("\n\nEnter request of process %d: ",(i+1));
for(j=0;j<no_r;j++)
{
printf("\n\tResource %d---- >",(j+1));
scanf("%d",&p[i].request[j]);
}
}
clrscr();
printf("\nInitially Available Resources: ");
resource_status(available);
process_status();
```

```
printf("\nCurrent Status of availability: ");
resource_status(avail);
k=detect();
if(k==0)
{
for(i=0;i<no_p;i++)
{
for(j=i+1;j<no_p;j++)
{
if(p[i].request[0]>p[j].request[0])
{
temp=p[i];
p[i]=p[j];
p[j]=temp;
}
}
}
k=detect();
}
getch();
}
void resource_status(int a[])
{
for(i=0;i<no_r;i++)
{
printf("\n\tResource %d---- >%d",(i+1),a[i]);
}
}
void process_status()
{
printf("\nPROCESS\t\tALLOCATION\tREQUEST");
```

```
for(i=0;i<no_p;i++)
{
printf("\n%s\t",p[i].pname);
printf("\t");
for(j=0;j<no_r;j++)
{
printf("%d ",p[i].allocation[j]);
}
printf("\t\t");
for(j=0;j<no_r;j++)
{
printf("%d ",p[i].request[j]);
}
}
}
int detect()
{
int work[10];int
x=0,y=0;
for(i=0;i<no_r;i++)
{
work[i]=avail[i];
}
for(i=0;i<no_p;i++)
{
for(j=0;j<no_r;j++)
{
if(p[i].allocation[j]!=0)
{
p[i].finish=0;
}
```

```
else
{
p[i].finish=1;
}
}
}
for(i=0;i<no_p;i++)
{
if(p[i].finish==0)
{
x=0;
for(j=0;j<no_r;j++)
{
if(p[i].request[j]<=work[j])
{
work[j]=work[j]+p[i].allocation[j];
x++;
}
if(x==no_r)
p[i].finish=1;
}
}
}
y=0;
for(i=0;i<no_p;i++)
{
if(p[i].finish==0)
{
printf("\nThe system is in deadlocked state because of the process %s",p[i].pname);
break;
}
```

```
else
{
y++;
}
}
if(y==no_p)
{
printf("\nThe system is in safe state.No deadlock");
printf("\nThe safe sequence is: ");
for(i=0;i<no_p;i++)
printf("%s ",p[i].pname);
return 1;
}
else
{
return 0;
}
}
```

SAMPLE INPUT AND OUTPUT:

Case1:

Enter number of processes: 5

Enter 1 process name: p0

Enter 2 process name: p1

Enter 3 process name: p2

Enter 4 process name: p3

Enter 5 process name: p4

Enter number of resources: 3

Enter availability of each resource.

Enter availability of 1 resource: 7

Enter availability of 2 resource: 2

Enter availability of 3 resource: 6

Enter allocated resource of process 1:

Resource 1 ---- >0

Resource 2 ---- >1

Resource 3 ---- >0

Enter allocated resource of process 2:

Resource 1 ---- >2

Resource 2 ---- >0

Resource 3 ---- >0

Enter allocated resource of process 3:

Resource 1 ---- >3

Resource 2 ---- >0

Resource 3 ---- >3

Enter allocated resource of process 4:

Resource 1 ---- >2

Resource 2 ---- >1

Resource 3 ---- >1

Enter allocated resource of process 5:

Resource 1 ---- >0

Resource 2 ---- >0

Resource 3 ---- >2

Enter request of each process:

Enter request of process 1:

Resource 1 ---- >0

Resource 2 ---- >0

Resource 3 ---- >0

Enter request of process 2:

Resource 1 ---- >2

Resource 2 ---- >0

Resource 3 ---- >2

Enter request of process 3:

Resource 1 ---- >0

Resource 2 ---- >0

Resource 3 ---- >0

Enter request of process 4:

Resource 1 ---- >1

Resource 2 ---- >0

Resource 3 ---- >0

Enter request of process 5:

Resource 1 ---- >0

Resource 2 ---- >0

Resource 3 ---- >2

Initially Available Resources:

Resource 1 ---- >7

Resource 2 ---- >2

Resource 3 ---- >6

PROCESS ALLOCATION REQUEST

p0 0 1 0 0 0 0

p1 2 0 0 2 0 2

p2 3 0 3 0 0 0

p3 2 1 1 1 0 0

p4 0 0 2 0 0 2

Current Status of availability:

Resource 1 ---- >0

Resource 2 ---- >0

Resource 3 ---- >0

The system is in safe state: No deadlock

The safe sequence is: p0 p1 p2 p3 p4

Case2:

Enter number of processes: 5

Enter 1 process name: p0

Enter 2 process name: p1

Enter 3 process name: p2

Enter 4 process name: p3

Enter 5 process name: p4

Enter number of resources: 3

Enter availability of each resource.

Enter availability of 1 resource: 7

Enter availability of 2 resource: 2

Enter availability of 3 resource: 6

Enter allocated resource of process 1:

Resource 1 ---- >0

Resource 2 ---- >1

Resource 3 ---- >0

Enter allocated resource of process 2:

Resource 1 ---- >2

Resource 2 ---- >0

Resource 3 ---- >0

Enter allocated resource of process 3:

Resource 1 ---- >3

Resource 2 ---- >0

Resource 3 ---- >3

Enter allocated resource of process 4:

Resource 1 ---- >2

Resource 2 ---- >1

Resource 3 ---- >1

Enter allocated resource of process 5:

Resource 1 ---- >0

Resource 2 ---- >0

Resource 3 ---- >2

Enter request of each process:

Enter request of process 1:

Resource 1 ---- >0

Resource 2 ---- >0

Resource 3 ---- >0

Enter request of process 2:

Resource 1 ---- >2

Resource 2 ---- >0

Resource 3 ---- >2

Enter request of process 3:

Resource 1 ---- >0

Resource 2 ---- >0

Resource 3 ---- >1

Enter request of process 4:

Resource 1 ---- >1

Resource 2 ---- >0

Resource 3 ---- >0

Enter request of process 5:

Resource 1 ---- >0

Resource 2 ---- >0

Resource 3 ---- >2

Initially Available Resources:

Resource 1 ---- >7

Resource 2 ---- >2

Resource 3 ---- >6

PROCESS ALLOCATION REQUEST

p0 0 1 0 0 0 0

p1 2 0 0 2 0 2

p2 3 0 3 0 0 1

p3 2 1 1 1 0 0

p4 0 0 2 0 0 2

Current Status of availability:

Resource 1 ---- >0

Resource 2 ---- >0

Resource 3 ---- >0

The system is in deadlocked state because of the process p2

RESULT:

Thus the C program for deadlock detection is written successfully and tested with varioussamples.

B) DEADLOCK AVOIDANCE

AIM:

To write a C program to implement banker's algorithm for deadlock avoidance.

CONCEPTS USED:

Terms Used:

- **Available:** A vector of length m indicates the number of available resources of each type. If Available[j] equals k, there are k instances of resource type Rj available.
- Max: An n x m matrix defines the maximum demand of each process. If M[i][j] equals k, then process Pi may request at most k instances of resource type Rj.
- **Allocation:** An n x m matrix defines the number of resources of each type currently allocated to each process. If Allocation[i][j] equals k, then process Pi is currently allocated k instances of resource type Rj.
- **Need:** An n x m matrix indicates the remaining resource need of each process. If Need[i][j] equals k, then process Pi may need k more instances of resource type Rij to complete its task. Note that Need[i][j] equals Max[i][j]-Allocation[i][j].

Algorithms:

Safety Algorithm:

1. Let Work and Finish be vectors of length m and n, respectively.

InitializeWork = Available and Fnish[i] = false for i = 0 , 1 , ..., n - l .

2. Find an i such that

bothFinish[i] ==false

Need <= Work

If no such i exists, go to step 4.

3. Work = Work +

Allocation,Finish[i] = true

Go to step 2.

4. If Finish[i] == true for all. i, then the system is in a safe state.

Resource Request Algorithm:

1. If Requesti <= Need, go to step 2. Otherwise, raise an error condition, since the processhas exceeded its maximum claim.

2. If Requesti <= Available, go to step 3. Otherwise, Pi must wait, since the resources are

not available.

3. The system pretend to have allocated the requested resources to process Pi by modifyingthe state as follows:

a. Available = Available - Request;

b. Allocation = Allocation + Request;

c. Needi = Needj - Request;

4. If the resulting resource-allocation state is safe, the transaction is completed, and processPi is allocated to its resources. However, if the new state is unsafe, then Pi must wait for Request Pi, and the old resource-allocation state is restored.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<dos.h>
#include<string.h>
struct process
{
char pname[15];
int max[10],allocation[10],need[10],finish;
};
int available[10],avail[10];
int no_p,no_r,i,j,x,k;
struct process p[10],temp,*temp1;
int request[10];
char c;
char p_req[10];int
safety();
void resource_status(int[]);
void process_status();
void resource_request(struct process *,int req[]);
void main()
{
```

```
clrscr();
printf("\nEnter number of processes: ");
scanf("%d",&no_p);
for(i=0;i<no_p;i++)
{
printf("\nEnter %d process name: ",(i+1));
scanf("%s",&p[i].pname);
}
printf("\nEnter number of resources: ");
scanf("%d",&no_r);
printf("\nEnter availability of each resource.");
for(i=0;i<no_r;i++)
{
printf("\n\tEnter availability of %d resource:
",(i+1));scanf("%d",&available[i]);
avail[i]=available[i];
}
printf("\nEnter the maximum need of resources for each process:
");for(i=0;i<no_p;i++)
{
printf("\n");
for(j=0;j<no_r;j++)
{
lab1:
printf("\n\tEnter max need of resource %d for %s: ",(j+1),p[i].pname);
scanf("%d",&p[i].max[j]);
if(p[i].max[j]>available[j])
{
printf("\n\tMaximum need is greater than available resource.
pleaseenter smaller value.");
goto lab1;
```

```
}
}
}
for(i=0;i<no_p;i++)
{
printf("\n\nEnter allocated resource of process %d: ",(i+1));
for(j=0;j<no_r;j++)
{
lab2:
printf("\n\tResource %d---- >",(j+1));
scanf("%d",&p[i].allocation[j]);
if(p[i].allocation[j]>avail[j])
{
printf("\n\tAllocated resource is greater than
availableresource.Please enter smaller amount.");
goto lab2;
}
p[i].need[j]=p[i].max[j]-p[i].allocation[j];
avail[j]=avail[j]-p[i].allocation[j];
}
}
clrscr();
printf("\nInitially Available Resources: ");
resource_status(available);
process_status();
printf("\nCurrent Status of availability: ");
resource_status(avail);
k=safety();
if(k==0)
{
for(i=0;i<no_p;i++)
```

```
{
for(j=i+1;j<no_p;j++)
{
if(p[i].need[0]>p[j].need[0])
{
temp=p[i];
p[i]=p[j];
p[j]=temp;
}
}
}
}
k=safety();
while(1)
{
printf("\n\nAny Request for resources?(y/n): ");
c=getche();
if(c=='n'||c=='N')
break;
printf("\nWhich process is requesting resources: ");
scanf("%s",&p_req);
for(i=0;i<no_p;i++)
{
x=strcmp(p_req,p[i].pname);
if(x==0)
{
temp1=&p[i];
break;
}
}
printf("\nEnter the request of %s: ",temp1->pname);
```

```
for(i=0;i<no_r;i++)
{
scanf("%d",&request[i]);
}
resource_request(temp1,request);
}
getch();
}
void resource_status(int a[])
{
for(i=0;i<no_r;i++)
{
printf("\n\tResource %d---- >%d",(i+1),a[i]);
}
}
void process_status()
{
printf("\nPROCESS\tMAXIMUM\tALLOCATION\tNEED")
;
for(i=0;i<no_p;i++)
{
printf("\n%s\t",p[i].pname);
for(j=0;j<no_r;j++)
{
printf("%d ",p[i].max[j]);
}
printf("\t");
for(j=0;j<no_r;j++)
{
printf("%d ",p[i].allocation[j]);
}
printf("\t\t");
```

```
for(j=0;j<no_r;j++)
{
printf("%d ",p[i].need[j]);
}
}
}
int safety()
{
int work[10];int
x=0,y=0;
for(i=0;i<no_r;i++)
{
work[i]=avail[i];
}
for(i=0;i<no_p;i++)
{
p[i].finish=0;
}
for(i=0;i<no_p;i++)
{
if(p[i].finish==0)
{
x=0;
for(j=0;j<no_r;j++)
{
if(p[i].need[j]<=work[j])
{
work[j]=work[j]+p[i].allocation[j];
x++;
}
if(x==no_r)
```

```
p[i].finish=1;
}
}
}
y=0;
for(i=0;i<no_p;i++)
{
if(p[i].finish==1)
y++;
}
if(y==no_p)
{
printf("\nThe system is in safe state");
printf("\nThe safe sequence is: ");
for(i=0;i<no_p;i++)
printf("%s ",p[i].pname);
return 1;
}
else
{
printf("\nThe system is not in safe state");
printf("\nThe sequence is: ");
for(i=0;i<no_p;i++)
printf("%s ",p[i].pname);
return 0;
}
}
void resource_request(struct process *p,int request[])
{
int k=0;
for(i=0;i<no_r;i++)
```

```
{
if(request[i]<=p->need[i])
{
if(request[i]<=avail[i])
{
k++;
}
}
}
if(k==no_r)
{
for(i=0;i<no_r;i++)
{
if(request[i]<=p->need[i])
{
if(request[i]<=avail[i])
{
avail[i]=avail[i]-request[i];
  p->allocation[i]=p-
  >allocation[i]+request[i]; p->need[i]=p-
  >need[i]-request[i];
}
}
}
clrscr();
printf("\nRequested resources are
allocated.");process_status();
resource_status(avail);
}
else
{
printf("\nRequested resources cannot be granted since the resources
```

are not available.");

}

}

SAMPLE INPUT AND OUTPUT:

Enter number of processes:5

Enter 1 process name: P0

Enter 2 process name: P1

Enter 3 process name: P2

Enter 4 process name: P3

Enter 5 process name: P4

Enter number of resources: 3

Enter availability of each resource.

Enter availability of 1 resource: 10

Enter availability of 2 resource: 5

Enter availability of 3 resource: 7

Enter the maximum need of resources for each process:

Enter max need of resource 1 for p0: 7

Enter max need of resource 2 for p0: 5

Enter max need of resource 3 for p0: 3

Enter max need of resource 1 for p1: 3

Enter max need of resource 2 for p1: 2

Enter max need of resource 3 for p1: 2

Enter max need of resource 1 for p2: 9

Enter max need of resource 2 for p2: 0

Enter max need of resource 3 for p2: 2

Enter max need of resource 1 for p3: 2

Enter max need of resource 2 for p3: 2

Enter max need of resource 3 for p3: 2

Enter max need of resource 1 for p4: 4

Enter max need of resource 2 for p4: 3

Enter max need of resource 3 for p4: 3

Enter allocated resource of process 1:

Resource 1 ---- >0

Resource 2 ---- >1

Resource 3 ---- >0

Enter allocated resource of process 2:

Resource 1 ---- >2

Resource 2 ---- >0

Resource 3 ---- >0

Enter allocated resource of process 3:

Resource 1 ---- >3

Resource 2 ---- >0

Resource 3 ---- >2

Enter allocated resource of process 4:

Resource 1 ---- >2

Resource 2 ---- >1

Resource 3 ---- >1

Enter allocated resource of process 5:

Resource 1 ---- >0

Resource 2 ---- >0

Resource 3 ---- >2

Initially Available Resources:

Resource 1 ---- >10

Resource 2 ---- >5

Resource 3 ---- >7

PROCESS MAXIMUM ALLOCATION NEED

p0 7 5 3 0 1 0 7 4 3

p1 3 2 2 2 0 0 1 2 2

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

Current Status of availability:

Resource 1 ---- >3

Resource 2 ---- >3

Resource 3 ---- >2

The system is not in safe state

The sequence is: p0 p1 p2 p3 p4

The system is in safe state

The safe sequence is: p3 p1 p4 p2 p0

Any Request for resources?(y/n):y

Which process is requesting resources: p1

Enter the request of p1: 1

0

2

Requested resources are allocated.

PROCESS MAXIMUM ALLOCATION NEED

p3 2 2 2 2 1 1 0 1 1

p1 3 2 2 3 0 2 0 2 0

p4 4 3 3 0 0 2 4 3 1

p2 9 0 2 3 0 2 6 0 0

p0 7 5 3 0 1 0 7 4 3

Resource 1 ---- >2

Resource 2 ---- >3

Resource 3 ---- >0

Any Request for resources?(y/n):y

Which process is requesting resources: p4

Enter the request of p4: 3

3

0

Requested resources cannot be granted since the resources are not

available.Any Request for resources?(y/n): n

RESULT: Thus the C program for deadlock avoidance is written successfully and tested withvarious samples.

Ex. No: 7 **FILE OPERATIONSDATE:**

AIM:

To write a shell script performs the file operations.

ALGORITHM:

1 Start
2 Display the menu and get the choice.
3 If choice is 1, get the file name and create a file.
4 If choice is 2, get the file name and show the file.
5 If choice is 3, get source and destination file name and copy it.
6 If choice is 4, get the source and new file name and rename it.
7 If choice is 5, get the file name and append the text.
8 If choice is 6, get the source and new file name and create a short cut link.
9 If choice is 7, get the file name and count the
words.10 If choice is 8, the exit.

SOURCE CODE:

```
ans='y'
while [ $ans=y -o $ans=y ]
do
clear
echo "              Menu"
echo "1.Create a File"
echo "2.Show File control"
echo "3.Copy a File"
echo "4.Rename a File"
echo "5.Append a File"
echo "6.Linking a file"
echo "7.No.of words in a
File"echo "8.Exit"
echo "Enter Your Choice"
read choice
case $choice
in

1 ) echo " Enter the file name"
  read a
  cat >
  $a
  ;;
2 ) echo " Enter the file name"
  read a
```

```
   cat $a
   read
   ;;
3 )  echo  " Enter source File Name"
   read a
   echo " Enter source File Name"
   read  b
   cp  $a
   $b
   ;;
4 )  echo  " Enter the source File Name"
   read a
   echo "Enter the source file Name"
   read b
   mv "$a"  "$b"
   echo "Rename Successful"
   ;;
5 ) echo  "Enter the File Name"
   read a
   cat >>  $a
   ;;
6 ) echo  "Enter the New File Name"
   read a
   echo "Enter the New File Name"
    read   b
    In  $a
    $b
    ;;
7 ) echo  "enter the File Name"
  read a
  echo "no of words"
  wc –w $a
  ;;
8 ) wish=" "
echo "Do you want to continue (y/n)"
read wish
  If [ $wish  = 'y' -o $wish = 'y'  ]
  then
        continue
   else
        echo "Thanks"
        exit
    fi
    ;;
   esac
   done
```

OUTPUT:

```
    menu
1   create a file
2   show file
3   copy a file
4   rename a file
5   append a file
6   linking a file
7   no of words
8   exit

enter ur choice
1
enter filename
teach

file
read
writ
e
do u want to
continuey
     menu
1   create a file
2   show a file
3   copy a file
4   rename a file
5   append a file
6   linking a file
7   no of words
8   exit

enter ur choice
2
Enter the filename
teach
file

read
writ
e
do u want to
continuey
```

```
    menu
1  create a file
2  show a file
3  copy a file
4  rename a file
5  append a file
6  linking a file
7  no of words
8  exit

Enter ur
choice3
enter the source filename
teach
enter destination
filenamestud
do u want to
continuey

    menu

1  create a file
2  show file
3  copy a file
4  rename a file
5  append a file
6  linking a file
7  no of words
8  exit

Enter ur
choice4

enter source
filenameTeach
enter new filename
rename successful

do u want to
continuey

    menu

1  create a file
2  show a file
```

```
3  copy a file
4  rename a file
5  append a file
6  linking a file
7  no of words
8  exit

Enter ur
choice5
Enter filename
Compsem
it
cse

do u want to
continuey

     menu
1  create a file
2  show file
3  copy a file
4  rename a file
5  append a file
6  linking a file
7  no of
words8 exit

Enter ur
choice2
enter the filename
stud
file
read
writ
e
do u want to
continuey

     menu

1  create a file
2  show file
3  copy a file
4  rename a file
5  append a file
6  linking a file
7  no of words
```

8 exit

Enter ur
choice6
enter old filename
compsem
enter new filename
teach

do u want to
continuey

menu
1 create a file
2 show file
3 copy a file
4 rename a
file5 append a
file6 linking a
file 7 no of
word
8 exit

Enter ur
choice7
enter filename
compsem
no of words
5 compsem
do u want to
continueY

menu
1 create a file
2 show file
3 copy a file
4 rename a
file5 append a
file6 linking a
file7 no of
words 8 exit

Enter ur
choice8

RESULT:

Thus the above shell script for file operations was executed successfully.

EX. NO : 8

DATE:

UNIX UTILITES

AIM:

To Write a Shell Script performs the basic UNIX utilities.

ALGORITHM:

1. Start.
2. Display the menu and get the choice.
3. If choice is 1, get the file name and perform head command.
4. If choice is 2, get the file name and perform tail command.
5. If choice is 3, get the source and perform cut command.
6. If choice is 4, get the source and destination file name and copy it.
7. If choice is 5, get the source and destination file name and join it.
8. If choice is 6, get the file name and show the difference.
9. If choice is 7, perform msg command to send and receive message.
10. If choice is 8, the exit.

SOURCE CODE:

```
ch=1
while test $ch -ne o
do
clear
echo “Menu “
echo “1.Head”
echo “2.Tail”
echo “3.cut”
echo ”4.Paste”
echo “5.Join”
echo ”6.Diff”
echo ”7.Msg”
echo “8.Exit”
echo “enter your choice”
read ch case
$ch in

1) echo “Enter  the file name”
read a
head -16 $aread
; ;

  2 ) echo “enter the file name”
```

```
read    a
tail    $a
read
; ;

   3 ) echo ”enter the file name”
Read a
cut -c -2 $a
read
; ;

   4 ) echo “enter first file name”
read a
echo ”enter second file name”
read b paste
$a $bread
    ; ;

   5 ) echo “enter the source file name”
read a
echo “enter destination file name”
read b paste
$a $bread
; ;

   6 ) echo “enter two files”
read a
read b ff
$a $bad
echo “enable /disable (y/n) message
adoptions”g  $k
[ $k = y -o $k = y ]
En

7) echo “Message  received”
else
echo “Message not received”
fi
read
; ;
```

```
8 ) echo “Thanks”
exit
; ;
esac
done
```

OUTPUT:

```
menu
1.head
2.tail
3.cut
4.paste
5.join
6.diff
7.msg
8.exit

enter ur  choice1

enter file name
compsem

file
read
writeit
cse

menu
1.head
2.tail
3.cut
4.paste
5.join
6.diff
7.msg
8.exit

enter ur  choice2

enter file name
teach
```

file
read
writeit
cse

menu
1. head
2. tail
3. cut
4. paste
5. join
6. diff
7. msg
8. exit

enter ur choice3
enter filename
teach

fi re
writ
cs

menu
1. head
2. tail
3. cut
4. paste
5. join
6. diff
7. msg
8. exit

enter ur choice4

enter first filename
comp sem
enter second filename
stud

file file
read read
write writeit
cse

menu
1. head
2. tail
3. cut
4. paste
5. join
6. diff
7. msg
8. exit

enter ur choice5

enter source filename
compsem
enter source filename
compsem
enter destination filename
teach
file file
read read
write writeit
it
cse cse

menu
1. head
2. tail
3. cut
4. paste
5. join
6. diff
7. msg
8. exit

enter ur choice6

```
 enter two filename
compsem
stud

4,5d3
< it
< cse
exp2 ,sh[51]: read : not found.

menu
1.head
2.tail
  3. cut
  4. paste
  5. join
  6. diff
  7. msg
  8. exit


enter ur choice7
enable/disable[y/n] message adaption
The current status is y
message recived

menu
  1. head
  2. tail
  3. cut
  4. paste
  5. joi
  n
  6.diff
  7.msg
  8.exit


enter ur choice8
```

RESULT:

Thus the above shell script for basic UNIX utilities was executed successfully.

EX. NO: 9
DATE:

SORTING OF 'N' NUMBERS USING AWK

AIM:

To write a shell script for arrange the numbers.

ALGORITHM:

1. Start.
2. Read n numbers and store them in array a (i.e a[1],a[2],.etc) initialize i=1
3. Display menu
    1. Ascending order
    2. Descending menu
    Read choice (say choice)
4. if choice=1 repeat step 5 until i=n
5. j=1 Repeat until j=n
    if
    a[i]>a[j]
    t=a[i]
    a[j]=t
6. if choice =2 repeat step 7 until i=n
7. j=1 Repeat until j=n
    if a[i] < a[j]
    t=a[i]
    a[i]=
    a[j]
    a[j]=t
8. Print the numbers.
9. Stop

SOURCE CODE:

```
ch=1
while test $ch -le 4
do
echo  "1.Ascending  order"
echo  "2.Descending  order"
echo "3.exit"
echo "enter your choice"
read ch case
$ch in
  1 ) awk 'BEGIN {
printf "enter the no of data"
getline n
printf "enter the element"
for ( i=0;;i<n;i++ )
```

```
{
getline s[i]
}
for ( i=0;i<n;i++ )
{
for ( j=i+1;j<n;j++ )
{
if ( s[i]>s[j] )
{
t=s[i]
s[i]=s[j]
s[j]=t
}
}
}
printf “Ascending order is “
for ( i=0;i<n;i++ )
printf ( “%d\n”,s[i] );
} ’ ;;
  2 ) awk ‘BEGIN{
printf “enter the elements”
for(i=0;i<n;i++)
{
getline s[i]
}
for(i=0;i<n;i++)
{
for(j=j+1;j<n;j++)
{
if(s[i] <s[j])
{
t=s[i]
s[i]=s[j]
s[j]=t
}
}
}
printf  ”Descending  order is”
for(i=0;i<n;I++)
printf “%d\n”,s[i]
}’ ; ;
  3 ) exit;
  ;esac
  done
```

OUTPUT:

1. Ascending
2. Descending
3. Exit

enter choice1

enter no of data 5
enter element
89
76
23
34
14

The ascending order is14
23
34
76
89

1. Ascending
2. Descending
3. Exit

Enter choice2

Enter element23
34
12
56
45

The Descending order is
56
45
34
23
12

1. Ascending
2. Descending
3. Exit

enter choice3

RESULT:

Thus the above shell script for sorting was executed successfully.

EX: NO: 10 **CALCULATE $^{N}C_{R}$ VALUE USING RECURSION**
DATE:

AIM:

To write a shell script performs $^{n}C_{r}$ calculation using recursion.

ALGORITHM:

1. Start.
2. Read values for n and r.
3. Pass the parameters n, r, (n-r) to the user defined factorial function and store the returnedvalues in nf, rf, nrf respectively.
4. Apply the following $^{n}C_{r}$
   formula:res=nf\(rf*nrf)
5. Print res.
6. Stop.

SOURCE CODE:

```
fact()
{
i=1
a=1
while [ $i -le $x ]
do
  a= ' expr  $a \*  $i '
   i= ' expr   $i  +  1 '
done
}
echo "Enter the N value:"
read n
echo "Enter the R value:"
read r
x=$n
fact
nf=$
a
x=$r
fact
rf=$a
x=' expr $n - $r '
fact
nrf=$
a
res=' expr $rf \* snrf '
res=' expr $nf / $res '
echo "the combination of $n C  $r is $res."
```

OUTPUT:

```
Enter the N value:
5
Enter the R value:
4

The combination of 5 c 4 is 5
```

RESULT:

Thus the shell script for above program was written and verified.

EX: NO: 11 DISPLAY THE NUMBERS BETWEEN 1 AND 9999 IN WORDS

DATE:

AIM:

To write a Shell script displays the numbers between 1 and 9999 in words.

ALGORITHM:

1. Start.
2. Read the number.
3. Separate the number and depending upon the position display the value in words.
4. Stop.

SOURCE CODE:

```
clear
echo " Enter any number between 1-9999;"
read n
n1=$n
r= 'expr$n/1000'
n=
'expr$n%1000'
case $r in
1) echo "one thousand"; ;
2) echo "two thousand"; ;
3) echo "three thousand"; ;
4) echo "four thousand"; ;
5) echo "five thousand"; ;
6) echo "six thousand"; ;
7) echo "seven thousand"; ;
8) echo "eight thousand"; ;
9) echo "nine thousand"; ;

esac
r='expr$n/100'
n='expr$n%100'
case $r in
1) echo "one hundred"; ;
2) echo "two hundred"; ;
3) echo "three hundred"; ;
4) echo "four hundred"; ;
5) echo "five hundred"; ;
6) echo "six  hundred"; ;
7) echo "seven hundred"; ;
8) echo "eight hundred"; ;
9) echo "nine hundred"; ;
```

```
esac
if [ $n –ne 0 ]
then echo
”and”fi
if [ $n –gt 20 ]
then
r=’expr$n/10’
n=’expr$n%10’
case $r in
2) echo”twenty”; ;
3) echo”thirty”; ;
4) echo”forty”; ;
5) echo”fifty”; ;
6) echo”sixty”; ;
7) echo”seventy”; ;
8) echo”eighty”; ;
9) echo”ninety”; ;
esac
fi
case $n  in
1)  echo”one”; ;
2)  echo”two”; ;
3)  echo”three”; ;
4)  echo”four”; ;
5)  echo”five”; ;
6)  echo”six”; ;
7)  echo”seven”; ;
8)  echo”eight”; ;
9)  echo”nine”; ;
10)  echo”ten”; ;
11)  echo”eleven”; ;
12)  echo”twelve”; ;
13)  echo”thirteen”; ;
14)  echo”fourteen”; ;
15)  echo”fifteen”; ;
16)  echo”sixteen”; ;
17)  echo”seventeen”; ;
18)  echo”eighteen”; ;
19)  echo”nineteen”; ;
20)  echo”twenty”; ;

esac
```

OUTPUT:

Enter any number between 1-9999:

818
eight hundred
and
eighteen

RESULT:

Thus the above shell script was executed and verified.

EX: NO: 12 **PALINDROME CHECKING**

DATE:

AIM:

To write a Shell script for Palindrome Checking.

ALGORITHM:

1. Start.
2. Read the string.
3. Check the string is palindrome or not using wc, cut and ne operations.
4. Stop.

SOURCE CODE:

```
echo "enter the string"
read str
len=`echo $str |wc -
c`while test $len -ne
0 do
temp=`echo $str | cut -c
$len`
revstr=${revstr}${temp}
len=`expr $len - 1`
done
echo "the reversed string is
$revstr"if test $str = $revstr
then echo "the given string is a palindrome"
else echo "the given string is not a
palindrome"fi
```

OUTPUT:

```
enter the string
malayalam
the reversed string is
malayalamthe given string is a
palindrome

enter the string
hello
the reversed string is olleh
the given string is not a palindrome
```

RESULT:

Thus the above shell script was executed and verified.