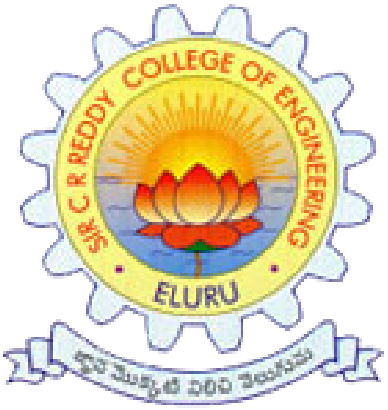
SIR CRR COLLEGE OF ENGINEERING, ELURU

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



II/IV B.Tech

OPERATING SYSTEMS LAB MANUAL CSE 2.2.8

FACULTY

G.NIRMALA

J.S.V.G.Krishna

CH.RAMADEVI

Operating Systems Laboratory

Index

CSE 2.2.8 OPERATING SYSTEMS LAB Credits:2

Instruction: 3 Hours Sessional Marks: 50

Univ. Exam : 3 Hours Univ-Exam-Marks:50

Course Objectives:

1) To learn about UNIX/LINUX operating system, its intervals.

2) To learn system programming for UNIX/LINUX Operating System.

3) To understand UNIX/LINUX shell and its programming.

4) To understand resource management policies and mechanisms and their performance evaluation.

Course Outcomes:

1) The student practices UNIX commands, Vi editor, shell commands.

2) The student develops skill in writing C programs using system calls for process management, Inter process communication and other aspects.

3) The student learns shell programming and develops skill for writing scripts for batch level tasks.

4) The student learns to simulate OS resource management aspects like process scheduling, Page Replacement and others to evaluate performance.

MODULE I

OS lab familiarization,

Home Assignment on Unix commands, Vi editor

Simple C programs using command line arguments, System calls, library function calls,

make utility.

C programs using fork system call to create process and study parent, child process

Mechanism

C programs to create process chaining, spawning

C programs to handle errors using errno, perror() function

C programs to use pipe system call for inter process communication

MODULE II

Familiarization of UNIX shell programming

Simple shell programming exercises

Shell programming using decision making constructs

Shell programming using loop constructs

Shell programming for file and directory manipulation

MODULE III

C programs to study process scheduling

FCFS

Shortest Job First

Round Robin

C programs to study page replacement

FIFO

Optimal

LRU page replacement

C programs to study deadlock avoidance and detection

C Programs to simulate free space management

References:

1. Unix concepts and applications by Sumitabha Das, TMH Publications.

2. Unix programming by Stevens, Pearson Education.

3. Shell programming by YashwanthKanetkar.

4. Operating System Concepts by Silberschatz, and Peter Galvin.

FEATURES OF UNIX

Some important features of UNIX are

- **Multiuser Capability**
- **Multitasking Capability**
- **Communication**
- **Security**
- **Portability**

Multiuser Capability: The process of performing more than one task at a time for more than one user is referred as multi user capability.

Multitasking Capability: The process of performing more than one task at a time for one user is called as multitasking capability.

Communication: Communication refers to the ability of one computer to communicate with other terminal to transfer data and/or programs.

Communications handled in UNIX are

- Communication between different terminals in the same terminal
- Communication between different terminals in the same location
- Communication between users and different terminals in
- Communication between computer of different types and sizes in different location.

SECURITY: Security projects one user from another and operating system from all other users. Security on UNIX systems is made available by

- Encryption Technologies of Files
- Providing Passwords
- Controlled Access of Individual File

PORTABILITY: Portability is the ability of the software to operate on different machines. UNIX runs on many types of computer than any other popular OS since it focuses mainly on the machine efficiency, most of the instructions are written in C and the remaining kernel program in machine language.

PROGRAMMING CAPABILITY: UNIX is a highly programmable OS .These program development tools assist user in reading, writing, executing and maintaining programs and it also includes compiler, assemblers and debuggers. Programming in UNIX is of two types:

- Shell programming
- Programming in language

SYSTEM TOOLS: System tools are programs that are not necessary to the computer's basic operation, but provide significant additional value for many applications.

The UNIX OS a collection of tools and also the user can create their own tools.

SYSTEMCALLS AND LIBRARIES: UNIX consists of frequency used programs that interacts with the kernel by invoking a well defined system calls which instructs the kernel to do various operations for the calling program and exchange data between the kernel and the program.

MODULE I

SIMPLE UNIX COMMANDS

1. **find**- finds directories

2. **cmp**- compares files & displays lines common to both

3. **diff**- compares files & displays lines different in both

4. **df**- frees disk spaces in blocks for root file system

5. **lp**- line printing

6. **date**- sets & displays date

7. **cal**- displays calendar

8. **ps**- reports process status

9. **kill**- terminates a specified process

10. **who**- lists all users logged presently

11. **finger**- displays user info if login is known

12. **man**- manual page like help

13. **write**- writes to another user terminal

14. **mesg**- allows device messages sent to a terminal

15. **mail**- communicates to other user

16. **vi**- visual editor for editing programs

17. **sh**- execute command for shell script file

18. **mkdir**- makes new directory

19. **rmdir**- removes directory

20. **mv**- moves files

21. **rm**- removes files

22. **cp**- copy source destination

23. **chmod**- change mode of file i.e. file permissions

24. **ls**-list of files

25. **grep**-globbally search the regular expression and print it

26. **echo-**displays the contents to the screen

27. **head-**displays the contents in forward direction

28. **tail-** displays the contents in reverse direction

29. **wc-** Count words, lines, and characters

30. **pwd**-displays the present working directory

31. **du filename** --- shows the disk usage of the files and directories in filename

32. **cd dirname** --- change directory.

33. **cat-**displays the contents of a file.

34. **Sort-**sort the file data

35. **Split-**splits file into smaller files.

36. **file**-determine the type of the file.

1.1 Program to print something using library functions and display ( ) function.

Aim: Write a sample program to display **UNIX programming lab** 'n' times using library function and display of int where n is an integer given to keyboard upon library function.

PROGRAM:

```
#include<stdio.h>
void display(int n);
main()
{
int n;
printf("enter the value of n");
scanf("%d",&n);
display(n);
}
void display(n)
{
int i;
for(i=1;i<=n;i++)
{
printf("UNIX programming lab\n");
}
}
```

Output:

```
Enter the value of  n
4
UNIX programming lab
UNIX programming lab
UNIX programming lab
UNIX programming lab
```

1.2. Program to display something using input and output files and system calls.

Aim: Write a sample program to display **UNIX programming lab** 'n' times where n is the input in in.txt file; the output will be displayed on out.txt file using system calls like fscanf and fprintf.

PROGRAM:

```
#include<stdio.h>
void display(int);
main()
{
int n;
FILE *fp;
fp=fopen("in.txt","r");
fscanf(fp,"%d",&n);
display(n);
fclose(fp);
}
void display(int n)
{
int i;
FILE *fp1;
fp1=fopen("out.txt","w");
for(i=1;i<=n;i++)
fprintf(fp1,"%s","\n UNIX programming lab");
fclose(fp1);
}
```

Output:

```
vi in.txt
3
./a.out
vi out.txt
UNIX programming lab
UNIX programming lab
UNIX programming lab
```

1.3. Program to implement fork system call

Aim: C programs using fork system call to create process and study parent, child process Mechanism

PROGRAM:

```
#include<stdio.h>
int main()
{
int i;
printf("hai before fork\n");
printf("i:%d\n",i);
i=fork();
printf("\n");
if(i==0)
{
printf("children started\n");
printf("children first time\n");
printf("getpid:%d\tgetppid:%d\n",getpid(),getppid());
sleep(5);
printf("child printing second time\n");
printf("getpid:%d\tgetppid:%d\n",getpid(),getppid());
}
else
{
printf("parent has started\n");
printf("getpid:%d\tgetppid:%d\n",getpid(),getppid());
printf("\n");
}
printf("hai after fork::%d\n",i);
return (0);
}
```

OUTPUT:

```
hai before fork
i:0
children started
children first time
getpid:22246    getppid:22245
parent has started
getpid:22245    getppid:22154
hai after fork::22246
[cs243@rhel4server ~]$ child printing second time
```

```
getpid:22246    getppid:12345
hai after fork::0
```

1.4. Program on error reporting using errno, perror( ) function

Aim: C programs to handle errors using errno, perror() function

PROGRAM:

```
#include<stdio.h>
#include<errno.h>
#include<stdlib.h>
extern int errno;
main()
{
FILE *fp;
fp=fopen("f1.c","r");
if(!fp)
{
terror();
printf("\n Error number is %d",errno);
printf("\n press enter \n");
getchar();
system("man perror");
}
else
printf("\n File already exists. \n");
fclose(fp);
}
terror()
{
printf("\n The file cannot be opened:");
printf("\n There is no such file (or) directory available");
}
```

Output:

The file cannot be opened
There is no such file (or) directory available
Error number is 02

1.5. Program on pipes

Aim: C programs to use pipe system call for inter process communication

PROGRAM:

```
#include<stdio.h>
#include<sys/types.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
 int pfds[2],i;
 char buf[25];
 pipe(pfds);
 if(!fork())
 {
  printf("CHILD : writing to pipe\n");
  write(pfds[1],"test",5);
  printf("CHILD: existing\n");
 }
 else
 {
  printf("PARENT: reading from pipe\n");
  read(pfds[0],buf,5);
  printf("PARENT: read %s \n",buf);
  wait(NULL);
 }
}
```

Output:

```
CHILD: writing to pipe
CHILD: existing
```

PARENT: reading from pipe
PARENT: read test

1.6. Program using make utility

Aim: C programs to add two numbers using make utility

vi a.c:

```
#include<stdio.h>

void add(int,int);

main()

{

int a,b;

printf("enter a,b values");

scanf("%d%d",&a,&b);

add(a,b);

}
```

vi b.c

```
void add(int x,int y)

{

int s;

s=x+y;

printf("sum of two numbers %d",s);

}
```

vi makefile:

```
add:a.o b.o

	gcc -o add a.o b.o

a.o:a.c
```

```
	gcc -c a.c

b.o:b.c

	gcc  -c b.c
```

out put:

//compailation process;

//$make

output

$./add
enter a,b values 2 3
sum of two numbers **5**

MODULE II

SHELL PROGRAMMING

Simple shell programming exercises

Shell programming using decision making constructs

2.1. To check the given no is even or odd

Aim: Write a shell program to find whether a given number is even or odd.

PROGRAM:

```
echo " Enter a number "
read n
let m=n%2
if [ $m -eq  0 ]
then
echo "Even number"
else
echo "Odd number"
fi
```

Output:

1) Enter a number: 4
   Even number

2) Enter a number: 9
   Odd number

2.2. Program to find greatest number using shell

Aim: Write a shell program to find the largest of three numbers.

PROGRAM:

```
echo " Enter three numbers "
read a b c
if [ $a -gt $b -a $a -gt $c ]
then
echo “$a is largest integer”
elif [ $b -gt $a –a $b –gt $c ]
then
echo “$b is largest integer”
elif [ $c –gt $a –a $c –gt $b ]
then
echo “$c is largest integer”
fi
```

Output:

```
Enter three numbers
4       6       9
9 is largest integer
```

Shell programming using loop constructs

2.3. To find factorial of a given number

Aim: Write a shell program to find the factorial of a given number.

PROGRAM:

```
let f=1, i=1
echo " Enter a number "
read n
while [ $i -le $n ]
do
let f=f*i
let i=i+1
done
echo " Factorial  of " $f
```

Output:

```
Enter a number: 6
Factorial of 6=720
```

2.4. Shell program to calculate the reverse of a given number

AIM: Write a shell program to calculate the reverse of a given number.

PROGRAM:

```
echo " Enter the number "
read n
let s=0
while [ $n -gt 0 ]
do
let m=n%10
let s=s*10+m
let n=n/10
done
echo "Reverse of the given number is " $s
```

Output:

```
Enter the number: 567
```

Reverse of the given number is 765

2.5. Shell program to find out the sum of the given number.
AIM: To write a shell program to find out the sum of the given number

PROGRAM:

```
echo "Enter the number:"
read n
let s=0
while test $n -gt 0
do
let m=n%10
let s=s+m
let n=n/10
done
echo " The sum of the given number is " $s
```

Output:

```
Enter the number:
33
The sum of the given number is = 6
```

2.6. Shell program for finding the given is whether an Armstrong number

AIM: Write a shell program to find the given number is an Armstrong number or not.
PROGRAM:

```
echo "Enter the number:"
read n
let s=0
let n1-n
while [ $n -gt 0 ]
do
let m=n%10
let s=s+m*m*m
let n=n/10
done
echo " S= " $s
if [ $s -eq $n1 ]
then
```

```
  echo " The number is an Armstrong number. "
else
  echo " The number is not an Armstrong number. "
fi
```

Output:

1) Enter the number:
   153

The number is an Armstrong number.

2) Enter the number:
   297

The number is not an Armstrong number.

2.7. Shell program for finding the given is whether a Palindrome number

AIM: Write a shell program to find the given number is a Palindrome or not.

PROGRAM:

```
Echo “Enter a number “
let s=0
read n
let n1=n
while [ $n –gt 0]
do
let m=n%10
let s= s*10+m
let n=n/10
done
if [ $s –eq $n1 ]
then
echo “The number is palindrome”
else
echo “The number is not palindrome”
fi
```

Output:

1) Enter a number 151

The number is palindrome

2) Enter a number 123
The number is not palindrome

2.8. Shell program for different linux commands and different arithmetic operators using case statement

AIM: Write a shell program to perform different linux commands using case statement

PROGRAM:

```
echo "1.ls 2.ps 3.username 4.calender 5.date "
while true
do
    echo "enter choice"
    read ch
    case "$ch" in
        1) echo "list of files"
            ls ;;
        2) echo "process status"
            ps ;;
        3) echo "username"
            uname ;;
        4) echo "calender"
           cal 7 2016 ;;
        5) echo "today date"
            date ;;
        *) exit
    esac
done
```

OUTPUT:

```
1.ls 2.ps 3.username 4.calender 5.date
enter choice
 1
list of files
adswit.sh  dead.c  fact.sh  fifo.c  fslab

enter choice
 2
process status
  PID TTY          TIME CMD
 4304 pts/1    00:00:00 bash
 4608 pts/1    00:00:00 sh
 4649 pts/1    00:00:00 cat
 4748 pts/1    00:00:00 sh
 4767 pts/1    00:00:00 sh
 4769 pts/1    00:00:00 ps
```

```
enter choice
3
username
Linux


enter choice
4
calender
     July 2016
Su Mo Tu We Th Fr Sa
             1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31

enter choice
5
today date
Wed Feb  8 10:30:19 IST 2017

enter choice
6
[cs3179@rhel4server oslab]$
```

AIM: Write a shell program to perform different arithmetic operators using Switch statement

PROGRAM:

```
                    echo " enter a value "
                     read a
                    echo " enter b value "
                     read b
                   echo "1.addition 2.subtract 3.multiplication 4.division  5.modular division"
                   while true
                   do
                      echo "enter the choice"
                      read ch
                     case "$ch" in
                        1) let  add=a+b
                           echo " addition of two numbers $add " ;;
                        2) let sub=a-b
                           echo " subtraction  of two numbers $sub " ;;
                        3) let mul=a*b
          echo " multiplication of two numbers $mul " ;;
       4) let div=a/b
```

```
        echo " division of two numbers $div " ;;
    5)  let mod=a%b
        echo " modular division of two numbers $mod" ;;
    *) exit
  esac
done
```

OUTPUT:

```
enter a value
5
enter b value
3
1.addition 2.subtract 3.multiplication 4.division 5.modular division
enter the choice
1
addition of two numbers 8

enter the choice
2
subtraction  of two numbers 2

enter the choice
 3
multiplication of two numbers 15

enter the choice
4
division of two numbers 1

enter the choice
5
modular division of two numbers 2

enter the choice
6
[cs3179@rhel4server oslab]$
```

MODULE III

C programs to study process scheduling

3.1. Write a C Program to simulate FCFS scheduling algorithm

Aim: To simulate FCFS algorithm when CPU burst time and arrival time are specified.

PROGRAM:

```
#include<stdio.h>
main()
{
  int bt[10],at[10],p[10],wt[10],tt[10];
  int i,n,sum=0,sum1=0;
  float avg1,avg2;
  printf("Enter the number of process:");
  scanf("%d",&n);
  for(i=0;i<n;i++)
   {
     printf("/n Enter the Burst time & Arrival time for %d processes:",i);
     scanf("%d",&bt[i]);
     scanf("%d",&at[i]);
   }
  for(i=0;i<n;i++)
   {
     if(i==0)
   {
     wt[i]=0;
   }
     else
   {
      sum1=sum1+bt[i-1];
```

```
        wt[i]=sum1-at[i];
        sum=sum+wt[i];
     }
  }
    avg1=sum/n;
  for(i=0;i<n;i++)
    {
        sum1=sum1+bt[i];
        tt[i]=sum1-at[i];
        sum=sum+tt[i];
    }
      avg2=sum/n;
printf("/n Process \t Burst time \t Arrival time \t Waiting time \t Turnaround time \t");
   for(i=0;i<n;i++)
      {
        printf("\n %d /t/t %d /t/t  %d /t/t %d /t/t  %d /t/t ",i+1, bt[i],at[i],wt[i],tt[i]);
      }
    printf("Average Waiting time=%f",avg1);
    printf("Average Turnaround time=%f",avg2);
  }
```

Output:

```
Enter the number of processes: 3
Enter the Burst time & Arrival time for 0 processes: 24   0
Enter the Burst time & Arrival time for 1 processes: 3      1
Enter the Burst time & Arrival time for 2 processes: 3      2

Process      Burst time      Arrival time    Waiting time   Turnaround time
1                  24              0              0              24
2                  3               1              23             26
3                  3               2              26             29

Average Waiting time= 16.333333
Average Turnaround time= 26.333333
```

3.2. Program to simulate SJF process scheduling algorithm

Aim: Program to simulate process scheduling like SJF.

PROGRAM:

```
#include<stdio.h>
main()
{
int p[5],w[5],b[5],t[5],i,j,n,t1,t2,avg,s[5];
float avg1,avg2,sum1=0.0,sum2=0.0;
printf("Enter value for n \n");
scanf("%d",&n);
printf("Enter burst time \n");
for(i=0;i<n;i++)
{
scanf("%d",&b[i]);
s[i]=b[i];
}
for(i=0;i<n;i++)
{
for(j=i+1;j<n;j++)
{
if(b[i]>b[j])
{
t1=b[i];
b[i]=b[j];
b[j]=t1;
}
}
}
w[0]=0;
t[0]=b[0];
for(i=1;i<n;i++)
{
w[i]=w[i-1]+b[i-1];
t[i]=t[i-1]+b[i];
}
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
if(b[i]==s[j])
{
printf("Waiting time of p[%d] is %d \n",i,w[i]);
printf("Turnaround time of p[%d]is %d \n",i,t[i]);
}
}
}
```

```
for(i=0;i<n;i++)
{
sum1=sum1+w[i];
sum2=sum2+t[i];
}
avg1=sum1/n;
avg2=sum2/n;
printf("Average Waiting time= %f \n",avg1);
printf("Average Turnaround time= %f \n",avg2);
}
```

Output:

```
[cs258@rhel4server ~]$ gcc sjf.c

[cs258@rhel4server ~]$ ./a.out


enter total number of processes3

enter burst time 24

3

3

enter the arrival time

0

1

2

process      burst time        arrival time     waiting time    turn around time


 p1          3                 1                0               3

 p2          3                 2                3               6
```

p3 24 0 6 30

the average waiting time is3.000000

average turn around timeis13.000000

3.3. Program to simulate RR process scheduling algorithm

Aim: Program to simulate process scheduling using Round Robin.

PROGRAM:

```
#include<stdio.h>
main()
{
  int st[10],bt[10],wt[10],tat[10],n,tq;
  int i,count=0,swt=0,stat=0,temp,sq=0;
  float awt=0.0,atat=0.0;
  printf("Enter number of process:");
  scanf("%d",&n);
  for(i=0;i<n;i++)
  {
   printf("Enter Burst time for p %d:",i+1);
   scanf("%d",&bt[i]);
   st[i]=bt[i];
  }
  printf("Enter Time Quantum:\n");
  scanf("%d",&tq);
  while(1)
  {
   for(i=0,count=0;i<n;i++)
   {
     temp=tq;
     if(st[i]==0)
     {
      count++;
      continue;
     }
     if(st[i]>tq)
     st[i]=st[i]-tq;
```

```
    else
    if(st[i]>=0)
    {
      temp=st[i];
      st[i]=0;
    }
    sq=sq+temp;
    tat[i]=sq;
   }
   if(n==count)
   break;
 }
 for(i=0;i<n;i++)
 {
  wt[i]=tat[i]-bt[i];
  swt=swt+wt[i];
  stat=stat+tat[i];
 }
 awt=(float)swt/n;
 atat=(float)stat/n;
 printf("process /t   Burst time /t  Waiting time /t  Turnaround time\n");
 for(i=0;i<n;i++)
 {
  printf("%d",i+1);
  printf("\t\t  %d",bt[i]);
  printf("\t   %d",wt[i]);
  printf("\t    %d",tat[i]);
  printf("\n");
 }
 printf("Average waiting time is:%f",awt);
 printf("\n Average turnaround time is:%f",atat);
}
```

Output:

```
[cs224@rhel4server ~]$ ./a.out


enter the number of processes3
```

enter the time quantum10

enter burst time for each job124

enter burst time for each job23

enter burst time for each job33

enter arrival time for each job 10

enter arrival time for each job 20

enter arrival time for each job 30

job interval burst completion remaining

1 0 24 10 14

2 10 3 13 0

3 13 3 16 0

1 16 14 26 4

1 26 4 30 0

turnarround time=59

average turn arround time=19.666666

waiting time=29

average waiting time +9.666667

3.4. Program to simulate Priority process scheduling algorithm

Aim: Program to simulate process scheduling using Priority.

PROGRAM:

```
#include<stdio.h>

int main()

{
   int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
   printf("Enter Total Number of Process:");
   scanf("%d",&n);
   printf("\nEnter Burst Time and Priority\n");
   for(i=0;i<n;i++)
   {
      printf("\nP[%d]\n",i+1);
      printf("Burst Time:");
      scanf("%d",&bt[i]);
      printf("Priority:");
      scanf("%d",&pr[i]);
      p[i]=i+1;          //contains process number
   }

   //sorting burst time, priority and process number in ascending order using selection sort

   for(i=0;i<n;i++)
   {
      pos=i;
      for(j=i+1;j<n;j++)
      {
         if(pr[j]<pr[pos])
```

```
            pos=j;
        }
        temp=pr[i];
        pr[i]=pr[pos];
        pr[pos]=temp;
     temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;

       temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }
    wt[0]=0;    //waiting time for first process is zero
    //calculate waiting time
    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];
        total+=wt[i];
    }
 avg_wt=total/n;      //average waiting time
    total=0;
  printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];     //calculate turnaround time
        total+=tat[i];
        printf("\nP[%d]\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }
  avg_tat=total/n;     //average turnaround time
    printf("\n\nAverage Waiting Time=%d",avg_wt);
    printf("\nAverage Turnaround Time=%d\n",avg_tat);
    return 0;
}
```

output:

```
Enter Total Number of Process:5
Enter Burst Time and Priority
P[1] Burst Time:10
```

Priority:3 [2]
Burst Time:1
Priority:1
P[3]
Burst Time:2
Priority:4
P[4]
Burst Time:1

Priority:5
[5]
Burst Time:5
Priority:2

Process	Burst Time	Waiting Time	Turnaround Time
P[2]	1	0	1
P[5]	5	1	6
P[1]	10	6	16
P[3]	2	16	18
P[4]	1	18	19

Average Waiting Time=8
Average Turnaround Time=12

C programs to study page replacement

3.5. program to simulate FIFO page replacement algorithm

Aim : Write a Program to simulate page replacement like FIFO.

PROGRAM:

```
#include<stdio.h>
 int str[30],sl;
 float placement(int);
  main()
  {
  int nfs,fs[10];
  int i,j,chk;
  float pfr[10];
  printf("enter the length of string");
 scanf("%d",&sl);
 printf("enter the string");
```

```
for(i=0;i<sl;i++)
scanf("%d",&str[i]);
printf("enter the list of frame sizes");
scanf("%d",&nfs);
printf("enter frame size:");
for(i=0;i<nfs;i++)
scanf("%d",&fs[i]);
for(i=0;i<nfs;i++)
{
printf("\n page replacement for frame size=%d\n",fs[i]);
pfr[i]=placement(fs[i]);
}
for(i=0;i<nfs;i++)
{
for(j=0,chk=0;j<nfs;j++)
{
if(fs[i]>fs[j])
chk=sl;
if(chk==1&&pfr[i]<pfr[j])
printf("anamoly occured between frame sizes%d&%d\n",fs[i],fs[j]);
}
}
}
float placement(int fs)
{
int i,j,p;
int f[10],t[10];
int par[10][20];
int pf=0;
float pfr;
for(i=0;i<fs;i++)
{
f[i]=-1;
t[i]=0;
}
for(i=0;i<sl;i++)
{
p=-1;
for(j=0;j<fs;j++)
if(str[i]==f[j])
break;
if(j!=fs)
continue;
```

```
for(j=0;j<fs;j++)
if(f[j]==-1)
{
p=j;
break;
}
if(p==-1)
p=greater(t,fs);
f[p]=str[i];
t[p]=0;
pf++;
for(j=0;j<fs&&j<=i;j++)
t[j]++;
for(j=0;j<fs;j++)
par[j][pf-1]=f[j];
}
for(i=0;i<fs;i++)
{
for(j=0;j<pf;j++)
printf("%5d",par[i][j]);
printf("\n");
pfr=((float)pf/sl)*100;
printf("no of page fault=%d",pf);
printf("\n page fault rate=%0.3f",pfr);
return pfr;
}
}
int greater(int t[100],int fs)
{
int i,j,c;
for(i=0;i<fs;i++)
{
c=0;
for(j=0;j<fs;j++)
if(t[i]>=t[j])
c++;
if(c==fs)
break;
}
return i;
}
```

Output:

```
enter the length of string 2
enter the string3
5
enter the list of frame sizes3
enter frame size:4
5
6
 page replacement for frame size=4
    3    3
 no of page fault=2
 page fault rate=100.000
 page replacement for frame size=5
    3    3
 no of page fault=2
 page fault rate=100.000
 page replacement for frame size=6
    3    3
 no of page fault=2
 page fault rate=100.000
```

3.6. program to simulate optimal page replacement algorithm

Aim : Write a Program to simulate optimal page replacement algorithm.

PROGRAM:

```
#include<stdio.h>
int str[30],sl;
float placement(int);
main()
{
  int nfs,fs[10];
  int i,j,chk;
  float pfr[10];
  printf("enter the length of string");
  scanf("%d",&sl);
  printf("enter the string:");
  for(i=0;i<sl;i++)
  scanf("%d",&str[i]);
  printf("enter the number of frames");
  scanf("%d",&nfs);
  printf("enter frame size:");
```

```
 for(i=0;i<nfs;i++)
 scanf("%d",&fs[i]);
 for(i=0;i<nfs;i++)
 {
  printf("\n page replacement for frame size=%d\n",fs[i]);
  pfr[i]=placement(fs[i]);
 }
 for(i=0;i<nfs;i++)
 for(j=0,chk=0;j<nfs;j++)
 {
  if(fs[i]>fs[j])
  chk=sl;
  if(chk==sl&&pfr[i]<pfr[j])
  printf("\n BELADY'S ANOMALY occured between frame sizes%d&%d\n",fs[i],fs[j]);
 }
}
float placement(int fs)
{
 int i,j,k,p;
 int f[10],t[10];
 int par[10][20];
 float pfr;
 int pf=0;
 for(i=0;i<fs;i++)
 f[i]=-1;
 for(i=0;i<sl;i++)
 {
  p=-1;
  for(j=0;j<fs;j++)
  t[j]=0;
  for(j=0;j<fs;j++)
  if(str[i]==f[j])
   break;
  if(j!=fs)
   continue;
  for(j=0;j<fs;j++)
  if(f[j]==-1)
  {
   p=j;
   break;
  }
  if(p==-1)
  {
```

```
    for(j=i+1;j<sl;j++)
    {
     for(k=0;k<fs;k++)
     if(t[k]==0 && str[j]==f[k])
     {
      t[k]++;
      break;
     }
     if(k!=fs)
     {
      for(k=0;k<fs;k++)
      if(t[k]!=0)
      t[k]++;
     }
   }
  p=lesser(t,fs);
 }
 f[p]=str[i];
 pf++;
 for(j=0;j<fs;j++)
 par[j][pf-1]=f[j];
}
for(i=0;i<fs;i++)
{
  for(j=0;j<pf;j++)
  printf("%5d",par[i][j]);
  printf("\n");
}
pfr=((float)pf/sl)*100;
printf("number of page faults=%d",pf);
printf("\n page fault rate =%0.3f\n",pfr);
return pfr;
}
int lesser(int t[10],int fs)
{
  int i,j,c;
  for(i=0;i<fs;i++)
  {
   c=0;
   for(j=0;j<fs;j++)
   if(t[i]<=t[j])
   c++;
   if(c==fs)
```

```
   break;
  }
 return i;
}
```

Output:

```
enter the length of string3
enter the string:2
3
4
enter the number of frames2
enter frame size:2 3
 page replacement for frame size=2
    2    2    4
   -1    3    3
 number of page faults=3
 page fault rate =100.000
 page replacement for frame size=3
    2    2    2
   -1    3    3
   -1   -1    4
 number of page faults=3
 page fault rate =100.000
```

3.7. program to simulate LRU page replacement algorithm

Aim : Program to simulate simulate page replacement like LRU

PROGRAM:

```
#include<stdio.h>
int str[30],sl;
float placement(int);
main()
{
int nfs,fs[30];
int i,j,chk;
float pfr[10];
printf("enter the length of the string:");
scanf("%d",&sl);
```

```
printf("enter the string:");
for(i=0;i<sl;i++)
scanf("%d",&str[i]);
printf("enter the no.of frame sizes");
scanf("%d",&nfs);
printf("enter the frame sizes");
for(i=0;i<nfs;i++)
scanf("%d",&fs[i]);
for(i=0;i<nfs;i++)
{
printf("\n page replacement for frame size=%d\n",fs[i]);
pfr[i]=placement(fs[i]);
}
for(i=0;i<nfs;i++)
for(j=0,chk=0;j<nfs;j++)
{
 if(fs[i]<fs[j])
chk=sl;
if(chk=sl&&pfr[i]<pfr[j])
printf("\n belady's anamoly occured between frame sizes%d&%d\n",fs[i],fs[j]);
}
}
float placement (int fs)
{
int i,j,k,p,f[10],t[10];
int par[10][20];
int pf=0;
float pfr;
for(i=0;i<fs;i++)
{
f[i]=-1;
t[i]=0;
}
for(i=0;i<sl;i++)
{
p=-1;
for(j=0;j<fs;j++)
if(str[i]==f[j])
break;
if(j!=fs)
continue;
for(j=0;j<fs;j++)
if(f[j]==-1)
```

```
{
p=j;
break;
}
if(p==-1)
{
for(j=0;j<i;j++)
{
for(k=0;k<fs;k++)
{
if(str[j]==f[k])
{
t[k]=0;
break;
}
}
for(k=0;k<fs;k++)
{
if(str[j]==f[k])
{
t[k]=0;
break;
}
}
for(k=0;k<fs&&k<=i;k++)
t[k]++;
}
p=greater(t,fs);
}
f[p]=str[i];
pf++;
for(j=0;j<fs;j++)
par[j][pf-1]=f[j];
}
for(i=0;i<fs;i++)
{
for(j=0;j<pf;j++)
  printf("%5d",par[i][j]);
  printf("\n");
}
pfr=((float)pf/sl)*100;
printf("no.of page faults=%d",pf);
printf("\n page fault rate=%0.3f",pfr);
```

```
return pfr;
}
 int greater(int t[10],int fs)
{
int i,j,c;
for(i=0;i<fs;i++)
{
c=0;
for(j=0;j<fs;j++)
if(t[i]>=t[j])
c++;
if(c==fs)
break;
}
return i;
}
```

Output:

```
enter the length of the string:3
enter the string: 4 5 6
enter the no.of frame sizes
3
enter the frame sizes
4 3 2

 page replecment for frame size=4
   4    4    4
  -1    5    5
  -1   -1    6
  -1   -1   -1
 no.of page faults=3
 page fault rate=100.000
 page replecment for frame size=3
   4    4    4
  -1    5    5
  -1   -1    6
 no.of page faults=3
 page fault rate=100.000
 page replecment for frame size=2
   4    4    6
  -1    5    5
```

no.of page faults=3
page fault rate=100.000

3.8 C programs to study deadlock avoidance and detection

```
/* Bankers Deadlock Avoidance Algorithm*/
include <stdio.h>
#include <stdlib.h>
int main()
{
   int Max[10][10], need[10][10], alloc[10][10], avail[10], completed[10], safeSequence[10];
   int p, r, i, j, process, count;
   count = 0;

   printf("Enter the no of processes : ");
   scanf("%d", &p);

   for(i = 0; i< p; i++)
      completed[i] = 0;

   printf("\n\nEnter the no of resources : ");
   scanf("%d", &r);

   printf("\n\nEnter the Max Matrix for each process : ");
   for(i = 0; i < p; i++)
   {
      printf("\nFor process %d : ", i + 1);
      for(j = 0; j < r; j++)
         scanf("%d", &Max[i][j]);
   }

   printf("\n\nEnter the allocation for each process : ");
   for(i = 0; i < p; i++)
   {
      printf("\nFor process %d : ",i + 1);
      for(j = 0; j < r; j++)
```

```
        scanf("%d", &alloc[i][j]);
}

printf("\n\nEnter the Available Resources : ");
for(i = 0; i < r; i++)
   scanf("%d", &avail[i]);

for(i = 0; i < p; i++)

   for(j = 0; j < r; j++)
      need[i][j] = Max[i][j] - alloc[i][j];

   do
   {
      printf("\n Max matrix:\tAllocation matrix:\n");

      for(i = 0; i < p; i++)
      {
         for( j = 0; j < r; j++)
            printf("%d ", Max[i][j]);
         printf("\t\t");
         for( j = 0; j < r; j++)
            printf("%d ", alloc[i][j]);
         printf("\n");
      }

      process = -1;

      for(i = 0; i < p; i++)
      {
         if(completed[i] == 0)//if not completed
         {
            process = i ;
            for(j = 0; j < r; j++)
            {
               if(avail[j] < need[i][j])
               {
                  process = -1;
                  break;
               }
            }
         }
         if(process != -1)
```

```
                break;
            }

            if(process != -1)
            {
                printf("\nProcess %d runs to completion!", process + 1);
                safeSequence[count] = process + 1;
                count++;
                for(j = 0; j < r; j++)
                {
                    avail[j] += alloc[process][j];
                    alloc[process][j] = 0;
                    Max[process][j] = 0;
                    completed[process] = 1;
                }
            }
        }
        while(count != p && process != -1);

        if(count == p)
        {
            printf("\nThe system is in a safe state!!\n");
            printf("Safe Sequence : < ");
            for( i = 0; i < p; i++)
                printf("%d ", safeSequence[i]);
            printf(">\n");
        }
        else
            printf("\nThe system is in an unsafe state!!");

}
```

output of the program is as follows:

```
Enter the no of processes : 5

Enter the no of resources : 3


Enter the Max Matrix for each process :
For process 1 : 7
```

```
5
3

For process 2 : 3
2
2

For process 3 : 7
0
2

For process 4 : 2
2
2

For process 5 : 4
3
3

Enter the allocation for each process :
For process 1 : 0
1
0

For process 2 : 2
0
0

For process 3 : 3
0
2

For process 4 : 2
1
1

For process 5 : 0
0
2

Enter the Available Resources : 3
3
2
```

```
 Max matrix:    Allocation matrix:
7 5 3          0 1 0
3 2 2          2 0 0
7 0 2          3 0 2
2 2 2          2 1 1
4 3 3          0 0 2

Process 2 runs to completion!
 Max matrix:    Allocation matrix:
7 5 3          0 1 0
0 0 0          0 0 0
7 0 2          3 0 2
2 2 2          2 1 1
4 3 3          0 0 2

Process 3 runs to completion!
 Max matrix:    Allocation matrix:
7 5 3          0 1 0
0 0 0          0 0 0
0 0 0          0 0 0
2 2 2          2 1 1
4 3 3          0 0 2

Process 4 runs to completion!
 Max matrix:    Allocation matrix:
7 5 3          0 1 0
0 0 0          0 0 0
0 0 0          0 0 0
0 0 0          0 0 0
4 3 3          0 0 2

Process 1 runs to completion!
 Max matrix:    Allocation matrix:
0 0 0          0 0 0
0 0 0          0 0 0
0 0 0          0 0 0
0 0 0          0 0 0
4 3 3          0 0 2

Process 5 runs to completion!
The system is in a safe state!!
Safe Sequence : < 2 3 4 1 5 >
```

VIVA QUESTIONS

1. **What is an OPERATING SYSTEM?**
2. **What are the various types of OPERATING SYSTEM**
3. **What are the basic functions of an operating system**
4. **What is multi tasking, multi programming, multi threading?**
5. **What is Dispatcher?**
6. **What is CPU Scheduler?**
7. **What is Context Switch?**
8. **What is Throughput, Turnaround time, waiting time and Response time?**
9. **Why paging is used?**
10. **What is the cause of thrashing? How does the system detect thrashing?**
11. **What is fragmentation? Different types of fragmentation?**
12. **What is DRAM? In which form does it store data?**

13. **What is cache memory?**

14. **What is a Safe State and what is its use in deadlock avoidance?**

15. **What is the difference between Hard and Soft real-time systems?**

16. **Explain Belady's Anomaly?**

17. **What is thrashing?**

18. **What are short, long and medium-term scheduling?**

19. **What are turnaround time and response time?**

20. **When is a system in safe state?**

21. **What is virtual memory?**

22. **What is kernel?**

23. **What is a thread?**

24. **What necessary conditions can lead to a deadlock situation in a system?**

25. **Briefly explain FCFS.**

26. **What is RR scheduling algorithm?**

27. **What is a Deadlock?**

28. **What is SJF scheduling algorithm?**

29. **What is priority scheduling algorithm?**

30. **What is a shell? various types of shells?**