COLLEGE OF ENGINEERING

THIRUVANANTHAPURAM

Operating Systems Lab

(CSL204)

Laboratory Record

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

COLLEGE OF ENGINEERING

THIRUVANANTHAPURAM

Operating Systems Lab

(CSL204)

YEAR .....

University Reg no.....

Name.....

Roll no..... Batch.....

Class..... from page No..... to page No.....

CERTIFIED BONAFIDE RECORD OF WORK DONE BY

.....

External Examiner Staff Member in-charge

Thiruvananthapuram

Date.....

Contents

1 EXPERIMENT 1

Aim

Getting started with linux basic commands for directory operations, displaying directory structure in tree format etc.

1.1 touch

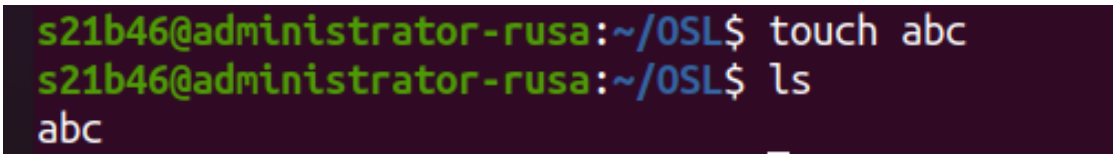
Purpose

`Used to make a new file in the current directory`

Usage

`touch filename`

Sample Input and Output



1.2 mkdir

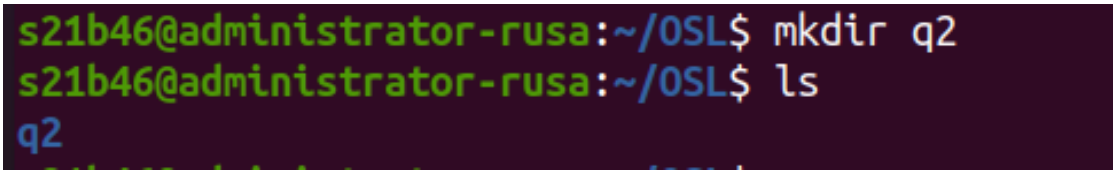
Purpose

`Used to create a new directory inside the current directory`

Usage

`mkdir filename`

Sample Input and Output



1.3 pwd

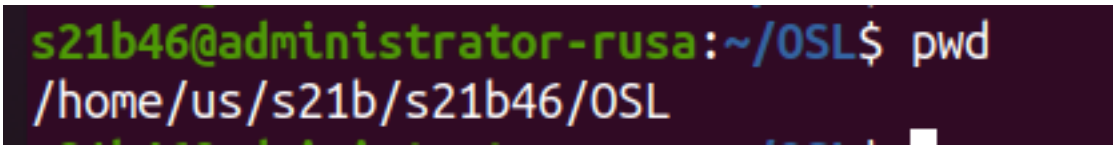
Purpose

Prints the present working directory

Usage

```
pwd
```

Sample Input and Output



1.4 cd

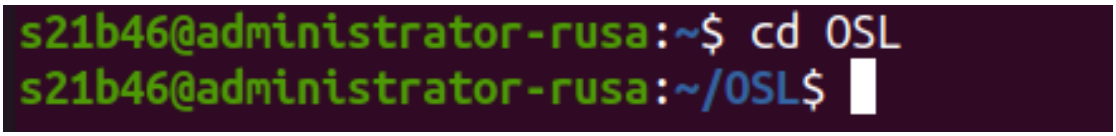
Purpose

Used to change directory

Usage

```
cd dirname
```

Sample Input and Output



1.5 cat

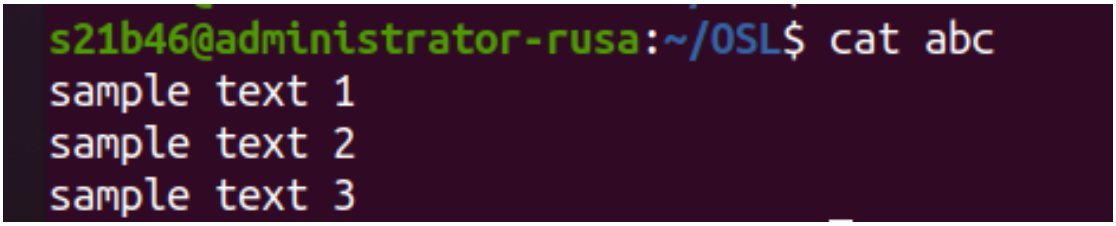
Purpose

Used to view content of a file

Usage

```
cat filename
```

Sample Input and Output



1.6 more

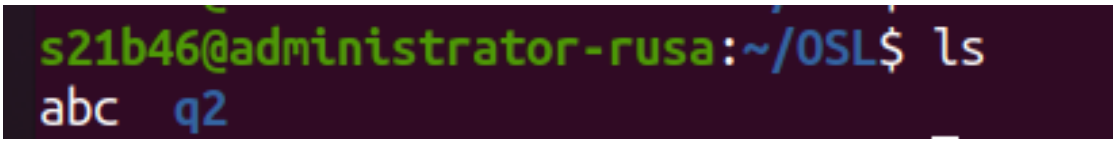
Purpose

```
Used to view content of a file one screenful at a time
```

Usage

```
more filename
```

Sample Input and Output



1.7 ls

Purpose

```
Used to display the files and subdirectories in a directory
```

Usage

```
ls
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL$ more abc
sample text 1
sample text 2
sample text 3
```

1.8 ls -l

Purpose

Provides long listing of all the files

Usage

```
ls -l
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL$ ls -l
total 8
-rw-rw-r-- 1 s21b46 s21b46   42 Mar  1 13:31 abc
drwxrwxr-x 2 s21b46 s21b46 4096 Mar  1 13:28 q2
```

1.9 ls -l -h

Purpose

Provides sizes of files in human readable form

Usage

```
ls -l -h
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL$ ls -l -h
total 8.0K
-rw-rw-r-- 1 s21b46 s21b46   42 Mar  1 13:31 abc
drwxrwxr-x 2 s21b46 s21b46 4.0K Mar  1 13:28 q2
```

1.10 ls -F

Purpose

Marks all the executable with * and directories with

Usage

```
ls -F
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL$ ls -F
abc  q2/
```

1.11 ls -a

Purpose

Show all the files in the present directory with special dot files

Usage

```
ls -a
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL$ ls -a
.  ..  abc  q2
```

1.12 cp

Purpose

Used to copy files and directories

Usage

```
cp filename1 filename2
cp -r dir1 dir2
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL$ cp abc q2
s21b46@administrator-rusa:~/OSL$ cd q2
s21b46@administrator-rusa:~/OSL/q2$ ls
abc
```

1.13 rm

Purpose

Used to remove a file

Usage

```
rm filename
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL/q2$ ls
abc  cde
s21b46@administrator-rusa:~/OSL/q2$ rm abc
s21b46@administrator-rusa:~/OSL/q2$ ls
cde
```

1.14 rmdir

Purpose

Used to remove a file

Usage

rm filename

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL$ ls
abc  q2
s21b46@administrator-rusa:~/OSL$ rmdir q2
s21b46@administrator-rusa:~/OSL$ ls
abc
```

1.15 clear

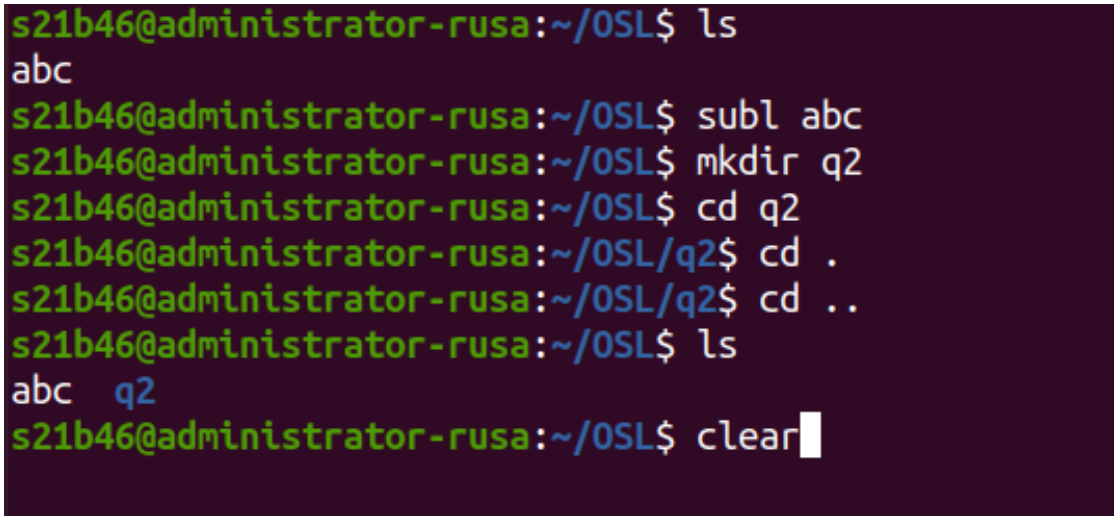
Purpose

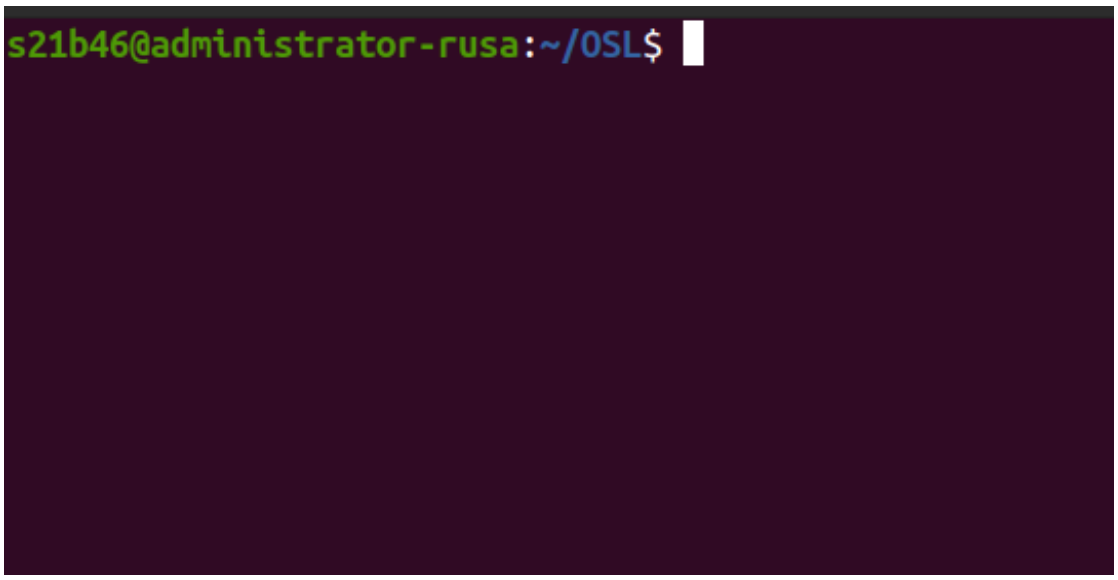
Clear the contents of the terminal.

Usage

clear

Sample Input and Output





1.16 man

Purpose

View help of the specified command name

Usage

man commandName

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL$ man clear
```

```
clear(1)                                General Commands Manual                                clear(1)

NAME
       clear - clear the terminal screen

SYNOPSIS
       clear [-Ttype] [-V] [-x]

DESCRIPTION
       clear  clears  your  screen  if this is possible, including its scrollback buffer (if the extended "E3" capability is
       defined).  clear looks in the environment for the terminal type given by the environment variable TERM, and  then  in
       the terminfo database to determine how to clear the screen.

       clear writes to the standard output.  You can redirect the standard output to a file (which prevents clear from actu-
       ally clearing the screen), and later cat the file to the screen, clearing it at that point.

OPTIONS
       -T type
            indicates the type of terminal.  Normally this option is unnecessary, because the  default  is  taken  from  the
            environment variable TERM.  If -T is specified, then the shell variables LINES and COLUMNS will also be ignored.

       -V   reports the version of ncurses which was used in this program, and exits.  The options are as follows:

       -x   do not attempt to clear the terminal's scrollback buffer using the extended "E3" capability.

HISTORY
       A clear command appeared in 2.79BSD dated February 24, 1979.  Later that was provided in Unix 8th edition (1985).

       AT&T  adapted a different BSD program (tset) to make a new command (tput), and used this to replace the clear command
       with a shell script which calls tput clear, e.g.,
 Manual page clear(1) line 1/81 43% (press h for help or q to quit)
```

1.17 tree

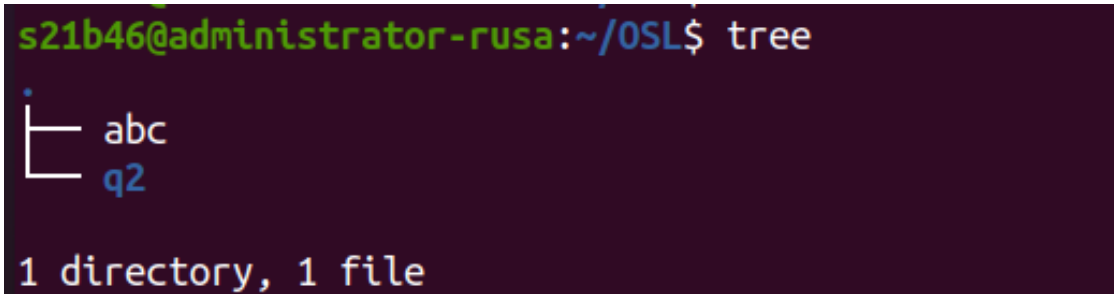
Purpose

Used to list or display the content of a directory in a tree-like format.

Usage

tree
tree dirname

Sample Input and Output



1.18 locate

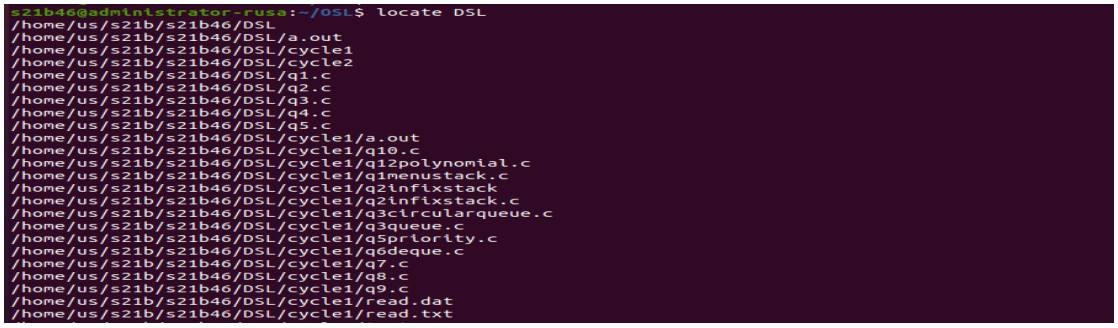
Purpose

Used to find files by their filename.

Usage

```
locate [options] filename
```

Sample Input and Output



1.19 kill

Purpose

Kill command is used to terminate process manually

Usage

kill processid

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

1.20 less

Purpose

Views content of a file one screenful at a time. less command is similar to the more command but faster than more.

Usage

less filename

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL$ less abc
s21b46@administrator-rusa:~/OSL$
```

```
sample text 1
sample text 2
sample text 3
abc (END)
```

1.21 who

Purpose

Used to display who is logged in.

Usage

who [options]

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL$ who
root     pts/0        2023-03-01 09:05 (192.168.99.249)
s21b51   pts/1        2023-03-01 13:11 (192.168.99.173)
s21b61   pts/2        2023-03-01 13:27 (192.168.99.195)
s21b41   pts/3        2023-03-01 13:13 (192.168.99.156)
s21b56   pts/4        2023-03-01 13:13 (192.168.99.168)
s20a35   pts/5        2023-03-01 09:11 (192.168.99.174)
s21b50   pts/6        2023-03-01 13:25 (192.168.99.174)
s21b74   pts/7        2023-03-01 13:14 (192.168.99.191)
s21b70   pts/8        2023-03-01 13:19 (192.168.99.178)
s21b42   pts/9        2023-03-01 13:13 (192.168.99.157)
s21b45   pts/10       2023-03-01 13:27 (192.168.99.159)
```

1.22 top

Purpose

Used to display the resources being used in your system.

Usage

top

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL$ top

top - 13:53:43 up  5:18, 44 users,  load average: 1.17, 1.28, 1.39
Tasks: 656 total,   1 running, 584 sleeping,   9 stopped,   0 zombie
%Cpu(s): 10.4 us,  7.0 sy,  0.0 ni, 82.4 id,  0.0 wa,  0.0 hi,  0.2 si,  0.0 st
KiB Mem : 65523032 total, 59592468 free,  2438636 used,  3491928 buff/cache
KiB Swap: 17674236 total, 17674236 free,        0 used. 62242976 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
29507 s20a69    20   0   31324    796    716 S  99.7  0.0 114:32.39 chatserver
31974 s21b46    20   0   42420   4376   3108 R   1.0  0.0   0:00.23 top
 1067 postgres  20   0  320116  26952  24944 S   0.7  0.0   0:36.98 postgres
 1431 postgres  20   0  323604  10888   5452 S   0.3  0.0   1:09.33 postgres
15775 s21b43    20   0   22792   5180   3428 S   0.3  0.0   0:00.18 bash
17882 s21b63    20   0   22792   5220   3468 S   0.3  0.0   0:00.22 bash
32065 s21b55    20   0    9856    948    844 S   0.3  0.0   0:00.01 pager
    1 root      20   0  226344  10092   6668 S   0.0  0.0   0:07.62 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.01 kthreadd
    4 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/0:0H
    6 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 mm_percpu_wq
    7 root      20   0       0      0      0 S   0.0  0.0   0:04.47 ksoftirqd/0
    8 root      20   0       0      0      0 I   0.0  0.0   0:17.76 rcu_sched
    9 root      20   0       0      0      0 I   0.0  0.0   0:00.00 rcu_bh
   10 root      rt   0       0      0      0 S   0.0  0.0   0:00.01 migration/0
   11 root      rt   0       0      0      0 S   0.0  0.0   0:00.03 watchdog/0
   12 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/0
   13 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/1
   14 root      rt   0       0      0      0 S   0.0  0.0   0:00.03 watchdog/1
   15 root      rt   0       0      0      0 S   0.0  0.0   0:00.09 migration/1
   16 root      20   0       0      0      0 S   0.0  0.0   0:01.26 ksoftirqd/1
   18 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/1:0H
   19 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/2
   20 root      rt   0       0      0      0 S   0.0  0.0   0:00.03 watchdog/2
```

1.23 chmod

Purpose

Used to modify file access rights.

Usage

```
chmod [options] permissions filename
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL$ chmod --help
Usage: chmod [OPTION]... MODE[,MODE]... FILE...
  or:  chmod [OPTION]... OCTAL-MODE FILE...
  or:  chmod [OPTION]... --reference=RFILE FILE...
Change the mode of each FILE to MODE.
With --reference, change the mode of each FILE to that of RFILE.

  -c, --changes          like verbose but report only when a change is made
  -f, --silent, --quiet  suppress most error messages
  -v, --verbose          output a diagnostic for every file processed
      --no-preserve-root  do not treat '/' specially (the default)
      --preserve-root    fail to operate recursively on '/'
      --reference=RFILE  use RFILE's mode instead of MODE values
  -R, --recursive        change files and directories recursively
      --help     display this help and exit
      --version  output version information and exit

Each MODE is of the form '[ugoa]*([-+=]([rwxXst]*|[ugo]))+|[-+=][0-7]+'.
```

1.24 chown

Purpose

Used to change user and/or group ownership of a file,directory or symbolic link.

Usage

```
chown [options] user[:group] filename
```

1.25 redirection (>)

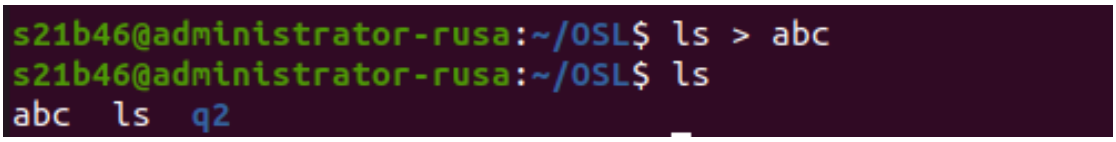
Purpose

Overwrites the file with the output of the command

Usage

```
command > filename
```

Sample Input and Output



1.26 redirection(»)

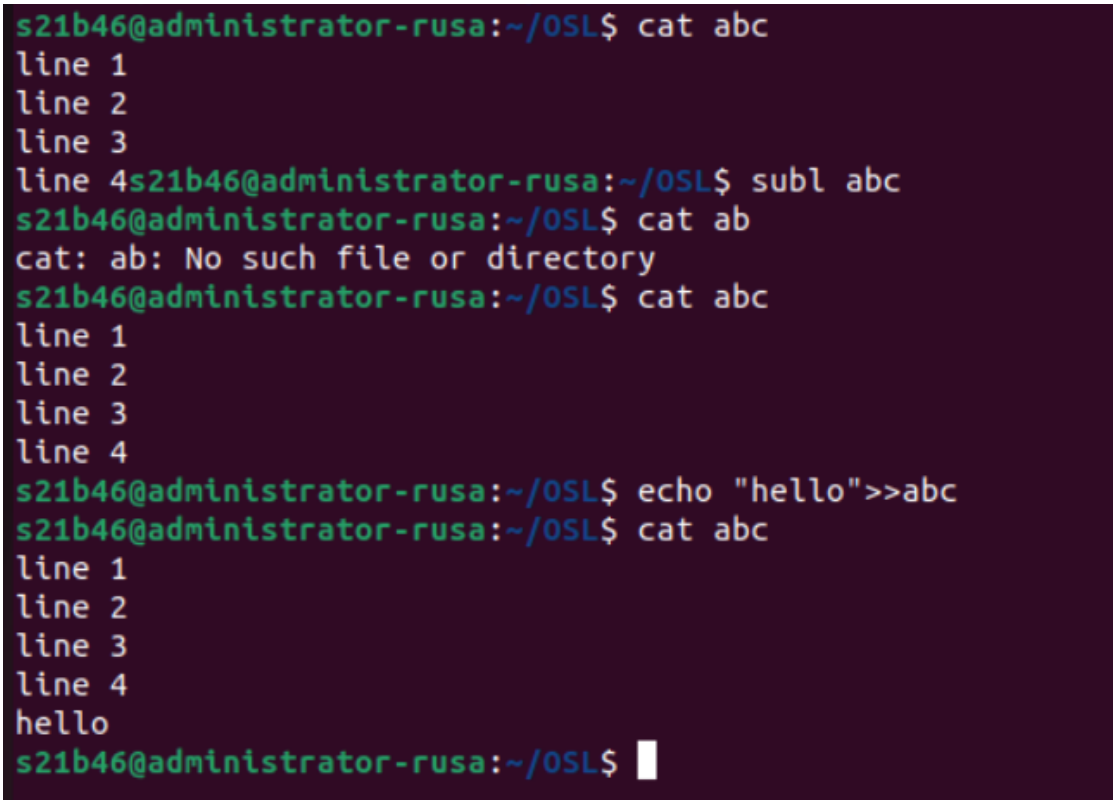
Purpose

Appends the file with the output of the command.

Usage

```
command » filename
```

Sample Input and Output



1.27 redirection(<)

Purpose

Used to redirect standard input to a command.

Usage

```
command < filename
```

Sample Input and Output

```
s21b46@administrator-rusa:~$ ls
Desktop  Documents  Downloads  Music  OSL  Pictures  Public  snap  Templates  Videos
s21b46@administrator-rusa:~$ touch file
s21b46@administrator-rusa:~$ ls > file
s21b46@administrator-rusa:~$ cat file
Desktop
Documents
Downloads
file
Music
OSL
Pictures
Public
snap
Templates
Videos
```

1.28 piping(|)

Purpose

Used to redirect standard output of one command to the standard input of another command.

Usage

```
command1 | command2
```

Sample Input and Output

```
s21b46@administrator-rusa:~$ ls
Desktop  Documents  Downloads  file  Music  OSL  Pictures  Public  snap  Templates  Videos
s21b46@administrator-rusa:~$ ls | head -3
Desktop
Documents
Downloads
s21b46@administrator-rusa:~$
```

1.29 Filters (sort)

Purpose

Sorts the standard input and sends the output to standard output.

Usage

```
sort filename
```

Sample Input and Output

```
s21b46@administrator-rusa:~/DSL$ sort q2.c
{
}
        {
        }
                {
                }

                                big[k]=teststr[k];
                                break;
        char s[25],big[25]=" ";
                char teststr[25]=" ";
                        cn=(int)(s[j])-96;
                        else
        for(i=0;i<n;i++)
                for(j=i;j<n;j++)
                        for(k=0;k<j;k++)
                if(strlen(teststr)>strlen(big))
                        if(testalph[cn]!=2)
#include <stdio.h>
#include <string.h>
        int i,cn,j,k;
        int n=strlen(s);
                int testalph[26]={0};
                k=0;
        printf("Biggest substring is: %s\n",big);
        printf("Enter string:");
        scanf("%s",s);
                        testalph[cn]++;
                                teststr[k++]=s[j];
void main()
```

1.30 Filters (uniq)

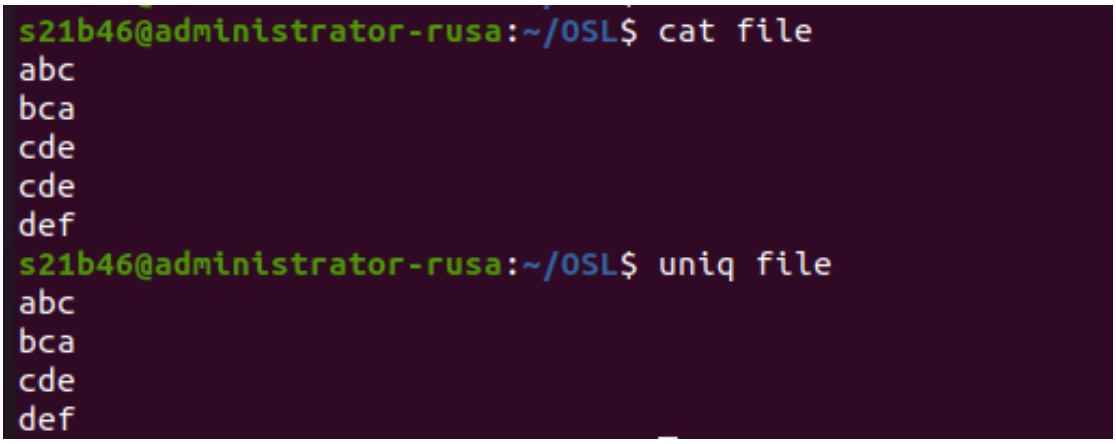
Purpose

Given a sorted stream of data from standard input it removes the duplicate lines of data and return the result to the standard output.

Usage

```
uniq filename
```

Sample Input and Output



1.31 Filters (grep)

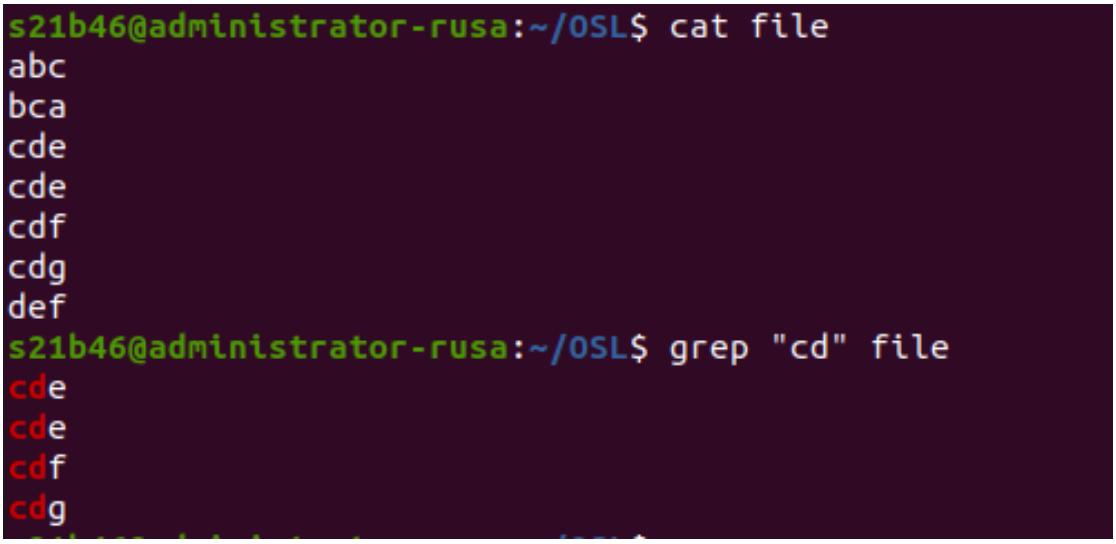
Purpose

Examines each line of data it recieves form standard input and outputs all lines that contains a specific pattern of characters.

Usage

```
grep ''string'' filename
```

Sample Input and Output



1.32 Filters (fmt)

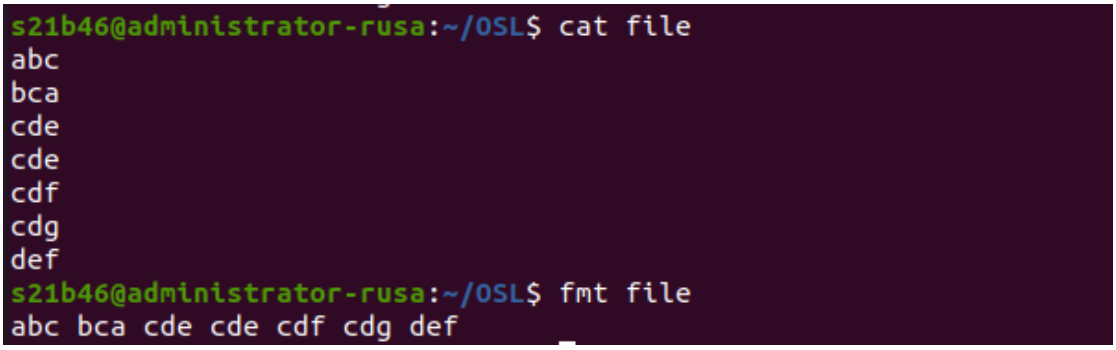
Purpose

Reads text from standard input and outputs formatted text to standard output

Usage

```
fmt filename
```

Sample Input and Output



1.33 Filters (pr)

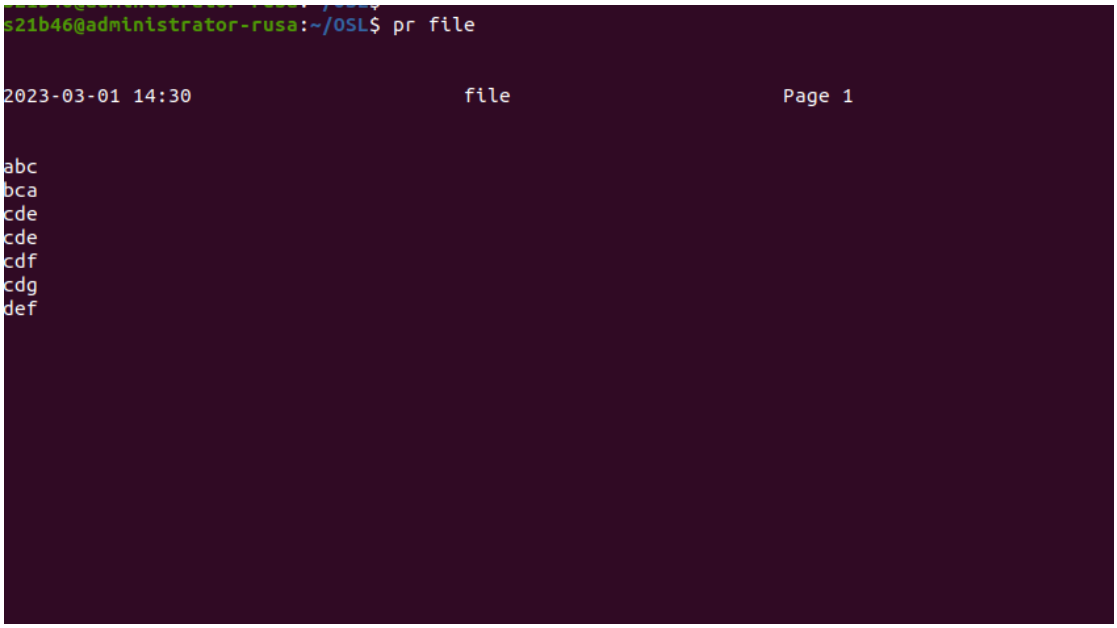
Purpose

Takes data from the standard input and splits data into pages with page breaks, footers and headers in preparation for printing

Usage

```
pr filename
```

Sample Input and Output



1.34 Filters (head)

Purpose

Outputs the first few lines of a file and returns it to the standard output

Usage

```
head -n filename (n is the number of lines to be printed, default value is 10)
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL$ head -3 file
abc
bca
cde
```

1.35 Filters (tail)

Purpose

Outputs the last few lines of a file and returns it to the standard output

Usage

tail -n filename (n is the number of lines to be printed, default value is 10)

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL$ tail -3 file
cdf
cdg
def
s21b46@administrator-rusa:~/OSL$
```

1.36 Filters (tr)

Purpose

Translate characters, can be used to perfrom tasks such as uppercase
to lowercase conversions

Usage

tr [: lower :] [: upper :]

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL$ tr [:lower:] [:upper:]
hello
HELLO
conversion
CONVERSION
```

1.37 Job Control (ps)

Purpose

List the processes running in the system.

Usage

ps

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL$ ps
  PID TTY          TIME CMD
17277 pts/31   00:00:00 bash
25278 pts/31   00:00:00 ps
s21b46@administrator-rusa:~/OSL$
```

1.38 su

Purpose

Temporarily become super user. It is used to switch from one user to another

Usage

su username

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL$ su
Password:
```

1.39 alias

Purpose

It lets the user to give names of his/her choice to a command or sequence of commands.

Usage

```
alias alternatename=command
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL$ alias listing=ls
s21b46@administrator-rusa:~/OSL$ listing
abc  file  q2
s21b46@administrator-rusa:~/OSL$
```

1.40 df

Purpose

The df command shows the size used space and available space on the mounted file system of your computer. Two of the most useful options are the -h (human readable) and -x (exclude) options. The human-readable option displays the sizes in Mb or Gb instead of in bytes.The exclude option allows you to tell df to discount filesystems you are not interested in.

Usage

```
df options filename
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL$ df
Filesystem      1K-blocks     Used  Available Use% Mounted on
udev             32737716        0   32737716   0% /dev
tmpfs             6552304     8828    6543476   1% /run
/dev/sda1      1132132716 62948572 1011648400   6% /
tmpfs            32761516    37564   32723952   1% /dev/shm
tmpfs                5120        0       5120   0% /run/lock
tmpfs            32761516        0   32761516   0% /sys/fs/cgroup
/dev/loop0          56960    56960          0 100% /snap/core18/2667
/dev/loop4           2688     2688          0 100% /snap/gnome-calculator/920
/dev/loop7          51072    51072          0 100% /snap/snapd/17950
/dev/loop10          1536     1536          0 100% /snap/gnome-system-monitor/181
/dev/loop3            768      768          0 100% /snap/gnome-logs/115
/dev/loop2          50816    50816          0 100% /snap/snapd/17883
/dev/loop17           768      768          0 100% /snap/gnome-characters/741
/dev/loop18         64896    64896          0 100% /snap/core20/1778
/dev/loop20           128      128          0 100% /snap/bare/5
/dev/loop5         354688   354688          0 100% /snap/gnome-3-38-2004/119
/dev/loop21           640      640          0 100% /snap/gnome-logs/112
/dev/loop6           2688     2688          0 100% /snap/gnome-system-monitor/178
/dev/loop23          2560     2560          0 100% /snap/gnome-calculator/884
/dev/loop8         354688   354688          0 100% /snap/gnome-3-38-2004/115
/dev/loop9          83328    83328          0 100% /snap/gtk-common-themes/1534
/dev/loop11         93952    93952          0 100% /snap/gtk-common-themes/1535
/dev/loop13        224256   224256          0 100% /snap/gnome-3-34-1804/77
/dev/loop12         64896    64896          0 100% /snap/core20/1822
/dev/loop16         56960    56960          0 100% /snap/core18/2679
/dev/loop14        463360   463360          0 100% /snap/gnome-42-2204/56
/dev/loop15         74752    74752          0 100% /snap/core22/504
/dev/loop1            512      512          0 100% /snap/gnome-characters/781
```

1.41 diff

Purpose

Comapares two text files and shows the difference between them. The -y (side by side) option shows lines, letting you focus on the lines which have differences.

Usage

```
diff options file1 file2
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL$ diff abc file
2,5c2,7
< ls
< q2
< abc
< q2
---
> bca
> cde
> cde
> cdf
> cdg
> def
```

1.42 echo

Purpose

echo command prints a string of text to the terminal window

Usage

echo ''string of text''

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL$ echo "Printing a senctence"
Printing a senctence
```

1.43 find

Purpose

find command is used to track down files that the user know exists but forgot its path.

Usage

find startlocation -name *filename*

Sample Input and Output

```
s21b46@administrator-rusa:~$ find OSL
OSL
OSL/abc
OSL/q2
OSL/file
```

1.44 free

Purpose

Gives a summary of the memory usage with the computer. -h option provides human friendly numbers and units.

Usage

free option

Sample Input and Output

```
s21b46@administrator-rusa:~$ free
               total        used        free      shared  buff/cache   available
Mem:        65523032     6402928    55346524      160924     3773580    58234436
Swap:       17674236           0    17674236
```

1.45 groups

Purpose

The group command tell which group the user is a member of.

Usage

groups username

Sample Input and Output

```
s21b46@administrator-rusa:~$ groups
s21b46
s21b46@administrator-rusa:~$ groups s21b46
s21b46 : s21b46
```

1.46 gzip

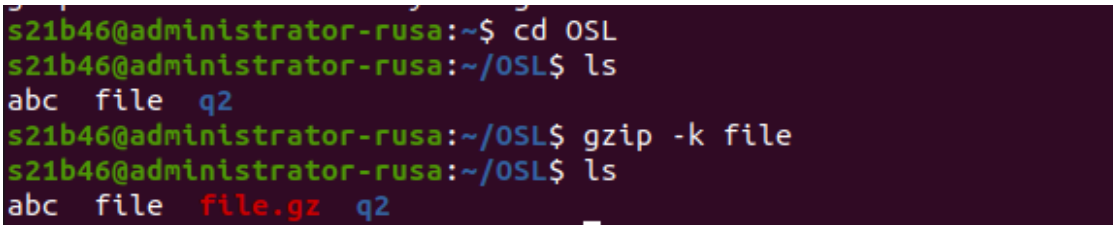
Purpose

gzip command compresses files. By default, it removes the original file and leaves you with the com

Usage

gzip option filename

Sample Input and Output



1.47 history

Purpose

The history command lists the commands you have previously issued on the command line. You can repe

Usage

history

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL$ history
  390  ./a.out
  391  cc q6deque.c
  392  ./a.out
  393  cc q6deque.c
  394  ./a.out
  395  cc q6deque.c
  396  ./a.out
  397  subl q7.c
  398  cc q7.c
  399  \
  400  ./a.out
  401  cc q7.c
  402  ./a.out
  403  cc q7.c
  404  ./a.out
  405  cc q7.c
  406  ./a.out
  407  cc q7.c
  408  ./a.out
  409  subl q9.c
  410  logout\
  411  logout
```

1.48 mc

Purpose

It is used to move files and directories from directory to directory

Usage

mv dir/file dirname (dir/file is the directory or file that is to be moved int dirname)

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL$ ls
abc  file  file.gz  q2
s21b46@administrator-rusa:~/OSL$ mv file.gz q2
s21b46@administrator-rusa:~/OSL$ ls
abc  file  q2
s21b46@administrator-rusa:~/OSL$ cd q2
s21b46@administrator-rusa:~/OSL/q2$ ls
file.gz
```

Result

Executed linux commands successfully.

2 EXPERIMENT 2(a) - Shell Programming Set-I

2.1 Question 1

Write a shell script that accepts one or more file name as arguments and convert the file contents to uppercase, provided they exist in the current directory.

Shell Script

```
for name in $@
do
echo "file name:"$name
tr [:lower:] [:upper:] < $name
echo " "
done
```

Sample Input and Output

```
s21b46@administrator-rusa:~/2a/q1$ cat c.txt
abc
hello
s21b46@administrator-rusa:~/2a/q1$ sh touppercase.sh c.txt
file name:c.txt
ABC
HELLO
```

2.2 Question 2

Write a shell script which accepts any number of arguments and prints them in the reverse order.

Shell Script

```
echo "no. of arguments:"$#
i=$#
echo -n "arguments in reverse order:"
while [ $i -ge 1 ]
do
echo -n ${!i}" "
i=$(($i-1))
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL/exp2/2a$ bash 2reverse.sh 1 2 3 4 5
no. of arguments:5
arguments in reverse order:5 4 3 2 1
s21b46@administrator-rusa:~/OSL/exp2/2a$
```

2.3 Question 3

Write a shell script that computes the gross salary of an employee according to the following rules (i) if basic salary is less than 1500 then HRA =10% of the basic and DA =90% of the basic. (ii) If basic salary is greater than or equal to 1500 then HRA =Rs.500 and DA=98% of the basic

Shell Script

```
echo "input basic salary:"
read base
if [ $base -lt 1500 ]
then
hra=$(echo "scale=2; $base*0.1" |bc)
da=$(echo "scale=2; $base*0.9" |bc)
gross=$(echo "scale=2; $base+$hra+$da" |bc)
else
hra=500
da=$(echo "scale=2; $base*0.98" |bc)
gross=$(echo "scale=2; $base+$hra+$da" |bc)
fi
echo "gross salary is":$gross
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL/exp2/2a$ bash 3salary.sh
input basic salary:
1500
gross salary is:3470.00
s21b46@administrator-rusa:~/OSL/exp2/2a$
```

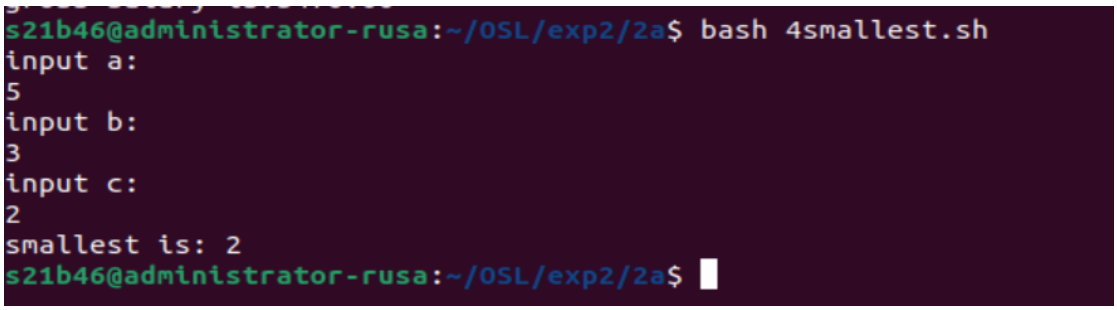
2.4 Question 4

Write a shell script to find smallest of 3 numbers that are read from keyboard.

Shell Script

```
echo "input a:"
read a
echo "input b:"
read b
echo "input c:"
read c
s=$a
if [ $b -lt $s ]
then
s=$b
fi
if [ $c -lt $s ]
then
s=$c
fi
echo "smallest is:"$s
```

Sample Input and Output



2.5 Question 5

Write a shell script to print a number in reverse order.

Shell Script

```
echo "input:"
read num
rev=0
dig=0
while [ $num -gt 0 ]
do
dig=$(($num % 10))
rev=$(( $rev*10 + $dig))
num=$(( $num/10 ))
done
echo "reverse is:"$rev
```

Sample Input and Output

```
s21b46@administrator-rusa:~/2a/q1$ cat c.txt
abc
hello
s21b46@administrator-rusa:~/2a/q1$ sh touppercase.sh c.txt
file name:c.txt
ABC
HELLO
```

2.6 Question 6

Write a shell script to print a number in reverse order.

Shell Script

```
echo "input lower bound:"
read a
echo "input upper bound:"
read b
i=$a
echo "the armstrong numbers between $a and $b are:"
while [ $i -le $b ]
do
num=$i
n=$num
power=0
while [ $n -gt 0 ]
do
power=$(($power+1))
n=$(($n/10))
done
n=$num
sum=0
while [ $n -gt 0 ]
do
dig=$(($n%10))
j=0
val=1
while [ $j -lt $power ]
do
val=$(($val*$dig))
j=$(($j+1))
done
sum=$(($sum+$val))
n=$(($n/10))
done
```

```
if [ $num -eq $sum ]
then
echo $num
fi
i=$(($i+1))
done
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL/exp2/2a$ bash 6armstrong.sh
input lower bound:
5
input upper bound:
1000
the armstrong numbers between 5 and 1000 are:
5
6
7
8
9
153
370
371
407
s21b46@administrator-rusa:~/OSL/exp2/2a$
```

2.7 Question 7

Write a shell program to print the following pattern upto n rows, for a given n.

```
1
2 2
3 3 3
4 4 4 4
.
.
n n n n n ..
```

Shell Script

```
echo "input n:"
read n
i=1
echo "pattern is:"
while [ $i -le $n ]
do
j=1
while [ $j -le $i ]
do
echo -n $i
j=$(($j+1))
```

```
done
echo "\n"
i=$(($i+1))
done
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL/exp2/2a$ bash 7pattern.sh
input n:
5
pattern is:
1
22
333
4444
55555
s21b46@administrator-rusa:~/OSL/exp2/2a$
```

EXPERIMENT 2(b) - Shell Programming Set-II

2.8 Question 1

Write a shell script to validate password strength. Here are a few assumptions for the password string. Length should be minimum of 8 characters. Should contain both small and capital case letters, atleast a digit and an underscore. If the password doesn't comply with any of theabove conditions, then the script should report it as a Weak Password.

Shell Script

```
echo "input string:"
read str
len=${#str}
if [ $len -ge 8 ]
then
if [[ $str =~ [A-Z] ]]
then
if [[ $str =~ [a-z] ]]
then
if [[ $str =~ [0-9] ]]
then
if [[ $str =~ "_" ]]
then
echo "Strong Password"
else
echo "Weak Password"
fi
else
echo "Weak Password"
fi
else
echo "Weak Password"
fi
else
echo "Weak Password"
fi
else
echo "Weak Password"
fi
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL/exp2/2b$ bash q1.sh
input string:
hello
Weak Password
s21b46@administrator-rusa:~/OSL/exp2/2b$ bash q1.sh
input string:
Hello123#@@_
Strong Password
s21b46@administrator-rusa:~/OSL/exp2/2b$
```

2.9 Question 2

Write a shell script to print the binary and hexadecimal equivalent of a decimal number.

Shell Script

```
echo "input decimal:"
read decimal
hex=$(echo "obase=16;ibase=10; $decimal"|bc)
binary=$(echo "obase=2;ibase=10; $decimal"|bc)
echo "hexadecimal:"$hex
echo "binary:"$binary
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL/exp2/2b$ bash q2.sh
input decimal:
0.634
hexadecimal :.A24
binary :.1010001001
s21b46@administrator-rusa:~/OSL/exp2/2b$ bash q2.sh
input decimal:
.32
hexadecimal :.51
binary :.0101000
```

2.10 Question 3

Write a shell script to generate all 3 digit numbers that contain only the digits 0, 1, 2, 3.(number does not start with 0)

Shell Script

```
i=100
while [ $i -lt 999 ]
do
n=$i
flag=1
while [ $n -gt 0 ]&&[ $flag -ne 0 ]
do
dig=$(($n%10))
if [ $dig -eq 0 ]||[ $dig -eq 1 ]||[ $dig -eq 2 ]||
[ $dig -eq 3 ]
then
n=$(($n/10))
else
flag=0
fi
done
if [ $flag -eq 1 ]
then
echo $i
fi
i=$(($i+1))
done
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL/exp2/2b$ bash q3.sh
 100
 101
 102
 103
 110
 111
 112
 113
 120
 121
 122
 123
 130
 131
 132
 133
 200
 201
 202
 203
 210
 211
 212
 213
 220
 221
 222
 223
 230
 231
 232
 233
 300
 301
 302
 303
 310
 311
 312
 313
 320
 321
 322
 323
 330
 331
 332
 333
s21b46@administrator-rusa:~/OSL/exp2/2b$
```

2.11 Question 4

Write a script to create the command ‘summarize directory’ which takes a directory as input and summarizes the contents in the directory. This command has to print the a. Total Number of Files in the directory b. List of all extensions of files present c. Count of files having each extension in new lines The command works recursively for all the subdirectories.

Shell Script

```
function summarize()
{
cd $1
result=${PWD##*/}
echo "Name of directory : $result"
files=$(find . -type f | wc -l)
echo -e "number of files : $files \n"
extensions=$(find . -type f | perl -ne 'print $1 if m/\.([^.\/]+)$/' | sort -u)
echo "List of all extensions of files present in directory"
echo "$extensions"
echo
echo "Extension and corresponding numbers "
for x in $extensions;do
length=$(find . -name "*.$x"| wc -l)
echo "number of $x files : $length"
done
}
```

Sample Input and Output

```
Name of directory : expt2b
number of files : 15

List of all extensions of files present in directory :-
png
sh
txt

Extension and corresponding numbers
number of png files : 7
number of sh files : 7
number of txt files : 1
```

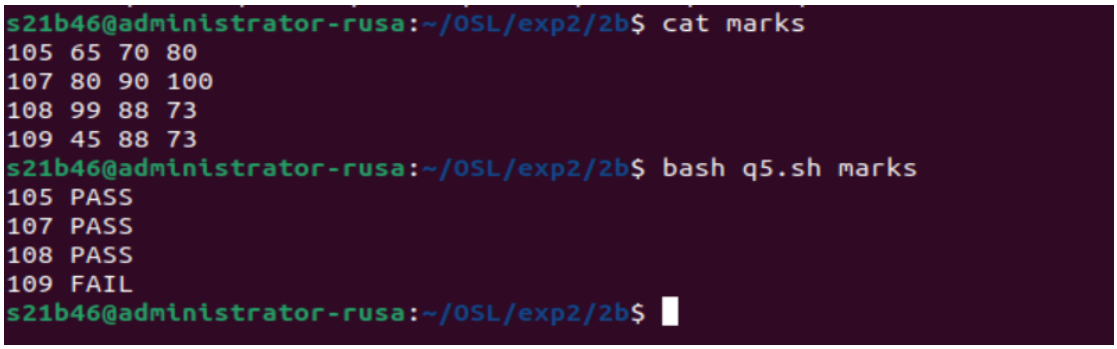
2.12 Question 5

Given a file containing the marks obtained by students for 3 subjects in an exam. In order to pass, student should score at least 50 marks in every subject. The file has one record(line) for each student in the following format: rollnumber subject1 subject2 subject3 Write a script to print pass/fail status of each student in the following format: rollnumber pass/fail.

Shell Script

```
file=$1
while read -r f1 f2 f3 f4
do
if [ $f2 -ge 50 ]&&[ $f3 -ge 50 ]&&[ $f4 -ge 50 ]
then
echo "$f1 PASS"
else
echo "$f1 FAIL"
fi
done<$file
```

Sample Input and Output



2.13 Question 6

Write a bash script to find the smallest prime number greater than n which is palindromic.

Shell Script

```
echo "input n:"
read n
num=$(($n+1))
flag=1
while [ $flag -ne 0 ]
do
i=2
prime=1
while [ $i -lt $num ]
do
rem=$(($num%$i))
if [ $rem -eq 0 ]
then
prime=0
i=$num
else
i=$((i+1))
fi
done
if [ $prime -eq 1 ]
then
m=$num
rev=0
while [ $m -gt 0 ]
do
dig=$(($m%10))
rev=$(( $rev*10 + $dig))
m=$(($m/10))
done
if [ $rev -eq $num ]
then
echo "smallest prime number greater than "$n" which is palindromic: " $num
flag=0
fi
fi
num=$(($num+1))
done
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL/exp2/2b$ bash q6.sh
input n:
313
smallest prime number greater than 313 which is palindromic:  353
s21b46@administrator-rusa:~/OSL/exp2/2b$
```

2.14 Question 7

Write a shell program to print the following pattern upto n rows, for a given n.

```
        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
  * * * * * * *
    * * * * *
      * * *
        *
```

Shell Script

```
echo "input n:"
read n
temp=$n
i=1
val=0
while [ $i -le $n ]
do
j=0
num=‘expr $n - $i‘
val=‘expr $num / 2‘
while [ $j -lt $val ]
do
echo -n " "
j=$(($j+1))
done
k=1
while [ $k -le $i ]
do
echo -n "*"
k=$(($k+1))
done
j=0
while [ $j -lt $val ]
do
echo -n " "
j=$(($j+1))
done
echo
i=$(($i+2))
done
i=$(($n-2))
while [ $i -gt 0 ]
do
j=0
num=‘expr $n - $i‘
```

```
val=‘expr $num / 2‘
while [ $j -lt $val ]
do
echo -n " "
j=$(($j+1))
done
k=1
while [ $k -le $i ]
do
echo -n "*"
k=$(($k+1))
done
j=0
while [ $j -lt $val ]
do
echo -n " "
j=$(($j+1))
done
echo
i=$(($i-2))
done
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL/exp2/2b$ bash q7.sh
input n:
5
  *
 ***
*****
 ***
  *
s21b46@administrator-rusa:~/OSL/exp2/2b$
```

2.15 Result

Executed shell programs successfully.

3 System calls of Linux operating system

3.1 System call: fork

System call description

fork() system call is used to create a new process

Program Code

```
#include<stdio.h>
#include<unistd.h>
int main()
{
int pid;
pid=fork();
if(pid>0)
printf("process %d creates child process %d\n",getpid(),pid);
return 0;
}
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL/exp3$ ./a.out
process 92576 creates child process 92577
s21b46@administrator-rusa:~/OSL/exp3$
```

3.2 System call: exec

System call description

exec() system call is used to replace the process's memory space with a new program

Program Code

```
#include<stdio.h>
#include<unistd.h>
int main()
{
int pid;
pid=fork();
if(pid>0)
printf("process %d creates child process %d\n",getpid(),pid);
return 0;
}
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL/exp3$ ./a.out
parent process 93728 executing
Hi

child process 93729 executing
calling program prg.c from child using exec
s21b46@administrator-rusa:~/OSL/exp3$
```

3.3 System call: getpid

System call description

getpid system call is used to get the process id
of a process

Program Code

```
#include<stdio.h>
#include<unistd.h>
void main()
{
printf("process running\n");
int pid=getpid();
printf("process id of running process:%d\n",pid);
}
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL/exp3$ ./a.out
process running
process id of running process:94880
s21b46@administrator-rusa:~/OSL/exp3$
```

3.4 System call: exit

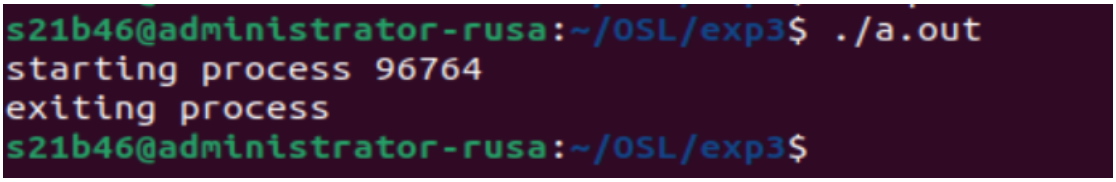
System call description

exit system call is used to terminate a process

Program Code

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
printf("starting process %d",getpid());
printf("\nexiting process\n");
exit(0);
printf("exited");
return(0);
}
```

Sample Input and Output



3.5 System call: wait

System call description

A parent process can wait for a child process to terminate using wait system call.

Program Code

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/wait.h>
int main()
{
int pid=fork(); //creating child
if(pid<0)
{
printf("fork failed!\n");
}
else if(pid==0)
{
printf("\nchild process %d executing\n",getpid());
printf("calling program prg.c from child using exec\n");
char* argv[]={"prg",NULL};
execv("./prg",argv);
}
else
{
printf("parent process %d executing\n",getpid());
int status;
printf("parent waiting for child to terminate\n");
int pid=wait(&status);
printf("child process %d exited-parent wait over\n",pid);
printf("parent process %d exits\n",getpid());
}
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL/exp3$ ./a.out
parent process 97555 executing
parent waiting for child to terminate

child process 97556 executing
calling program prg.c from child using exec
child process 97556 exited-parent wait over
parent process 97555 exits
```

3.6 System call: close

System call description

A close system call is a system call used to close a file descriptor by the kernel. For most file systems, a program terminates access to a file in a filesystem using the close system call.

Program Code

```
#include<stdio.h>
#include<fcntl.h>
#include<unistd.h>
#include<stdlib.h>
void main()
{
int fd=open("a.txt",O_RDONLY);
if(fd<0)
{
exit(0);
}
else
{ printf("file a.txt opened with file descriptor %d\n",fd);
char str[15];
int len=read(fd,str,10);
str[len]='\0';
printf("data read from a.txt:%s\n",str);
int e=close(fd);
if(e<0)
printf("close failed!\n");
else
printf("closed the file a.txt with file descriptor %d\n",fd);
}
}
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL/exp3$ ./a.out
file a.txt opened with file descriptor 3
data read from a.txt:hello
closed the file a.txt with file descriptor 3
s21b46@administrator-rusa:~/OSL/exp3$
```

3.7 System call: stat

System call description

Stat system call is a system call in Linux to check the status of a file such as to check when the file was accessed. The stat() system call actually returns file attributes. The file attributes of an inode are basically returned by Stat() function. An inode contains the metadata of the file. An i node contains: the type of the file, the size of the file, when the file was accessed (modified, deleted) that is time stamps, and the path of the file, the user ID and the group ID, links of the file, and physical address of file content.

Program Code

```
#include<stdio.h>
#include<sys/stat.h>
void main()
{
struct stat sfile;
stat("a.txt",&sfile);
printf("size of file a.txt: %ld\n",sfile.st_size);
printf("mode of file a.txt: %o\n",sfile.st_mode);
printf("id of device in which a.txt resides: %ld\n",sfile.st_dev);
}
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL/exp3$ ./a.out
size of file a.txt: 5
mode of file a.txt: 100664
id of device in which a.txt resides: 2051
s21b46@administrator-rusa:~/OSL/exp3$
```

3.8 System call: opendir

System call description

The opendir() system call opens a directory stream corresponding to the directory name, and returns
pointer to the directory stream.

Program Code

```
#include<stdio.h>
#include<dirent.h>
void main()
{
DIR* dr=opendir("dir");
if(dr==NULL)
printf("could not open directory\n");
else
printf("directory 'dir' opened\n");
}
```

Sample Input and Output

```
s21b46@administrator-rusa:~/OSL/exp3$ ./a.out
directory 'dir' opened
```

3.9 System call: readdir

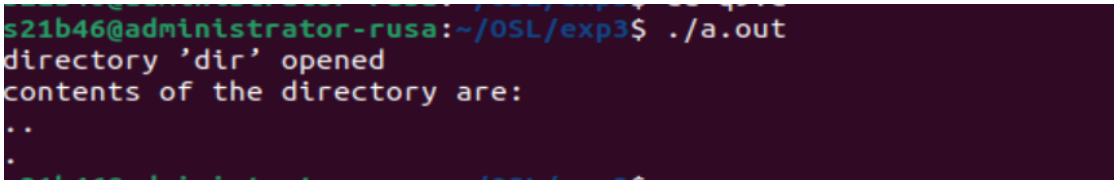
System call description

The readdir() system call function is used to read into a directory.The function returns a pointer

Program Code

```
#include<stdio.h>
#include<dirent.h>
void main()
{
DIR* dr=opendir("dir");
if(dr==NULL)
printf("could not open directory\n");
else
printf("directory 'dir' opened\n");
}
```

Sample Input and Output



3.10 Result

Executed System Call programs successfully.

4 EXPERIMENT 4-I/O System Calls

Aim

Write a program to use the following I/O system calls: open, close, read and write

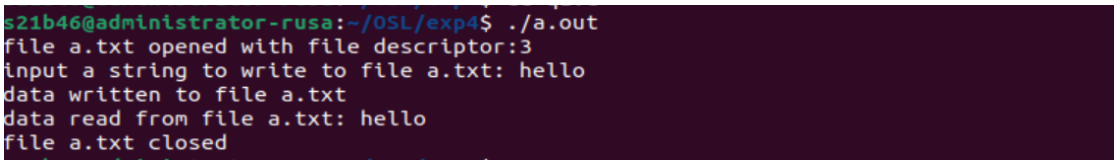
4.1 System call description

open() :
* A call to it creates a new open file descriptor, and is used to open a file.
* The open() function returns a new file descriptor. Unsuccessful attempt returns -1.
close() :
* It is used to close the corresponding file descriptor.
* Successful operation returns 0 and error returns -1.
read() :
* Used to retrieve data from a file stored in a file system.
* The value returned is the number of bytes read (or -1 for
error) and the file position is moved forward by this number.
write() :
* It writes data from a buffer to a file/device.
* The value returned is the number of bytes written (or -1 for error)successfully.

4.2 Program Code

```
#include<stdio.h>
#include<fcntl.h>
#include<unistd.h>
#include<stdlib.h>
#include<string.h>
void main()
{
int fd=open("a.txt",O_WRONLY|O_CREAT);
printf("file a.txt opened with file descriptor:%d",fd);
char str[100];
printf("\ninput a string to write to file a.txt: ");
scanf("%s",str);
write(fd,str,strlen(str)); //write to file
printf("data written to file a.txt");
close(fd);
fd=open("a.txt",O_RDONLY);
char bfr[100];
int len=read(fd,bfr,10);
bfr[len]='\0';
printf("\ndata read from file a.txt: %s",bfr);
close(fd);
printf("\nfile a.txt closed\n");
}
```

4.3 Sample Input and Output



4.4 Result

Executed file allocation strategies successfully.

5 EXPERIMENT 5: Inter-Process Communication using Shared Memory

5.1 Aim

Write a program to use the following I/O system calls: open, close, read and write

5.2 Algorithm

1. Start
2. Generate a unique key using ftok() function
3. Create a shared memory segment using shmget() function with key, size, and flags
4. Attach the process to the shared memory segment using shmat() function
5. Get the address of the shared memory segment and print it
6. Prompt the user to enter data to write to shared memory
7. Read the data entered by the user into a buffer
8. Copy the data from the buffer to the shared memory using strcpy() function
9. Print the data written to the shared memory
10. Detach the process from the shared memory segment using shmdt() function
11. Stop

Algorithm for reading from shared memory:

1. Start
2. Generate a unique key using ftok() function
3. Get the identifier of the existing shared memory segment using shmget() function with key, size, and flags
4. Attach the process to the shared memory segment using shmat() function
5. Get the address of the shared memory segment and print it
6. Read the data from the shared memory into a buffer
7. Print the data read from the shared memory
8. Detach the process from the shared memory segment using shmdt() function
9. Destroy the shared memory segment using shmctl() function with IPC_RMID flag
11. Stop

5.3 Program Code

Program for writing to shared memory

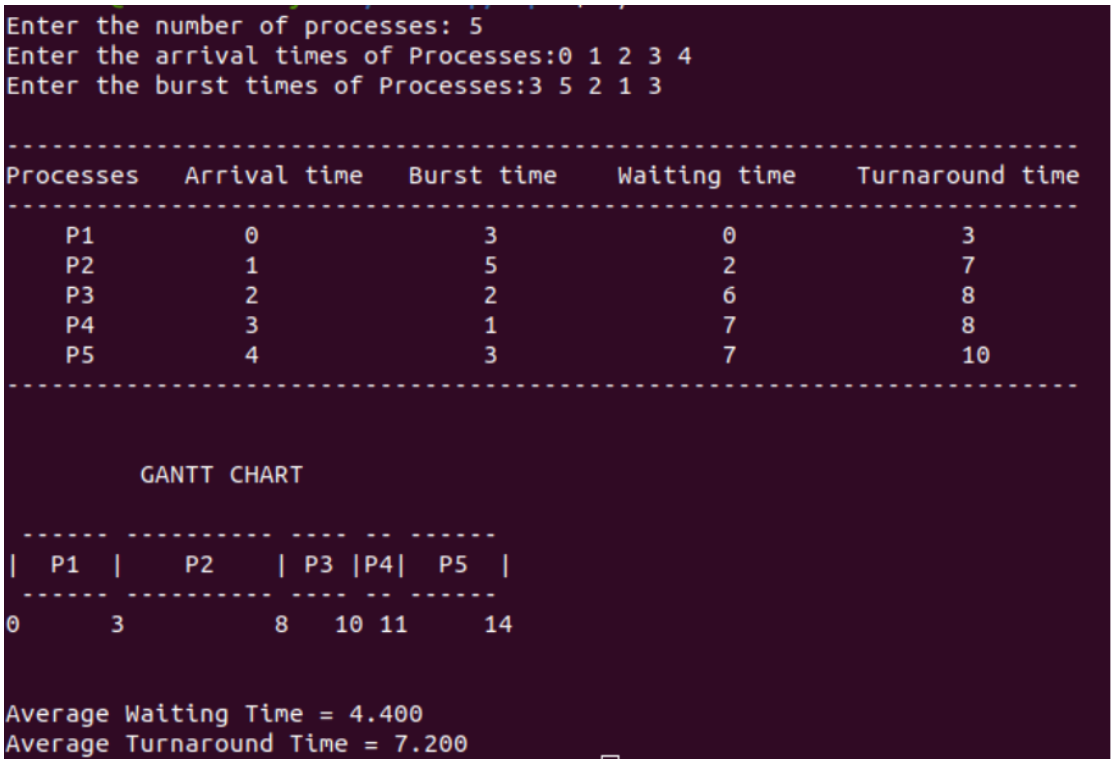
```
#include<sys/ipc.h>
#include<sys/shm.h>
#include<stdio.h>
#include<string.h>
int main()
{
key_t key=ftok("shmfile",43); // ftok to generate unique key
int shmid=shmget(key,1024,0666|IPC_CREAT);
printf("identifier for shared memory is %d\n",shmid);
void* shared_memory=shmat(shmid,NULL,0); //process attached to shared memory segment
printf("Process attached at %p\n",shared_memory);
printf("Enter data to write to shared memory:");
```

```
char buff[100];
scanf("%s",buff);
strcpy(shared_memory,buff); //data written to shared memory
printf("Data written to shared memory is:%s\n",(char *)shared_memory);
shmdt(shared_memory); //detach from shared memory
return 0;
}
```

Program for reading from shared memory

```
#include<sys/ipc.h>
#include<sys/shm.h>
#include<stdio.h>
#include<string.h>
int main()
{
key_t key=ftok("shmfile",43);
int shmid=shmget(key,1024,0666);
printf("identifier for shared memory is: %d\n",shmid);
void* shared_memory=shmat(shmid,NULL,0);
printf("Process attached at %p\n",shared_memory);
char buff[100];
printf("Data read from shared memory is:%s\n",(char *)shared_memory);
shmdt(shared_memory);
shmctl(shmid,IPC_RMID,NULL); //destroy shared memory
}
```

5.4 Sample Input and Output



5.5 Result

Executed Inter Process Communication Algorithms successfully.

6 Implement Dining Philosophers problem using Semaphores

6.1 Dining Philosopher's Problem

Aim

Implement a program for solving Dining Philosophers problem using Semaphores.

6.1.1 System call description

1. Start
2. Declare and initialize the semaphores for the chopsticks.
3. Create 5 threads with the 'pthread_create' function, and pass the philosopher function as an argument along with the index of the philosopher as an argument.
4. Within the philosopher function, wait for the left chopstick to become available by calling 'sem_wait' on the semaphore corresponding to the left chopstick.
5. Wait for the right chopstick to become available by calling 'sem_wait' on the semaphore corresponding to the right chopstick.
6. Sleep for one second to simulate eating.
7. Release the right chopstick by calling 'sem_post' on the semaphore corresponding to the right chopstick.
8. Release the left chopstick by calling 'sem_post' on the semaphore corresponding to the left chopstick.
9. In the main function, wait for all threads to complete using the 'pthread_join' function.
10. Stop

6.1.2 Program Code

```
#include<stdio.h>
#include<pthread.h>
2
#include<semaphore.h>
#include<unistd.h>
sem_t chopsticks[5];
void* philosopher(void* n)
{
int i=*(int*)n;
sem_wait(&chopsticks[i]); //wait
printf("philosopher %d picks up left chopstick\n",i);
sem_wait(&chopsticks[(i+1)%5]);
printf("philosopher %d picks up right chopstick\n",i);
printf("philosopher %d is eating\n",i);
sleep(1);
printf("philosopher %d has finished eating\n",i);
sem_post(&chopsticks[(i+1)%5]); //signal
printf("philosopher %d puts down right chopstick\n",i);
```

```
sem_post(&chopsticks[i]);
printf("philosopher %d puts down left chopstick\n",i);
printf("philosopher %d is thinking\n",i);
}
void main()
{
printf("no. of philosophers:5\n");
int arg[5];
pthread_t t[5];
for(int i=0;i<5;i++)
{
sem_init(&chopsticks[i],0,1);
}
for(int i=0;i<5;i++)
{ arg[i]=i;
pthread_create(&t[i],NULL,philosopher,(void*)&arg[i]);
}
for(int i=0;i<5;i++)
pthread_join(t[i],NULL);
}
```

6.1.3 Sample Input and Output

```
no. of philosophers:5
philosopher 0 picks up left chopstick
philosopher 1 picks up left chopstick
philosopher 1 picks up right chopstick
philosopher 1 is eating
philosopher 4 picks up left chopstick
philosopher 3 picks up left chopstick
philosopher 1 has finished eating
philosopher 1 puts down right chopstick
philosopher 1 puts down left chopstick
philosopher 1 is thinking
philosopher 2 picks up left chopstick
philosopher 0 picks up right chopstick
philosopher 0 is eating
philosopher 0 has finished eating
philosopher 0 puts down right chopstick
philosopher 0 puts down left chopstick
philosopher 0 is thinking
philosopher 4 picks up right chopstick
philosopher 4 is eating
philosopher 4 has finished eating
philosopher 4 puts down right chopstick
philosopher 4 puts down left chopstick
philosopher 4 is thinking
philosopher 3 picks up right chopstick
philosopher 3 is eating
philosopher 3 has finished eating
philosopher 3 puts down right chopstick
philosopher 3 puts down left chopstick
philosopher 3 is thinking
philosopher 2 picks up right chopstick
philosopher 2 is eating
philosopher 2 has finished eating
philosopher 2 puts down right chopstick
philosopher 2 puts down left chopstick
philosopher 2 is thinking
```

6.2 Producer Consumer Problem Using Semaphore

6.2.1 Aim

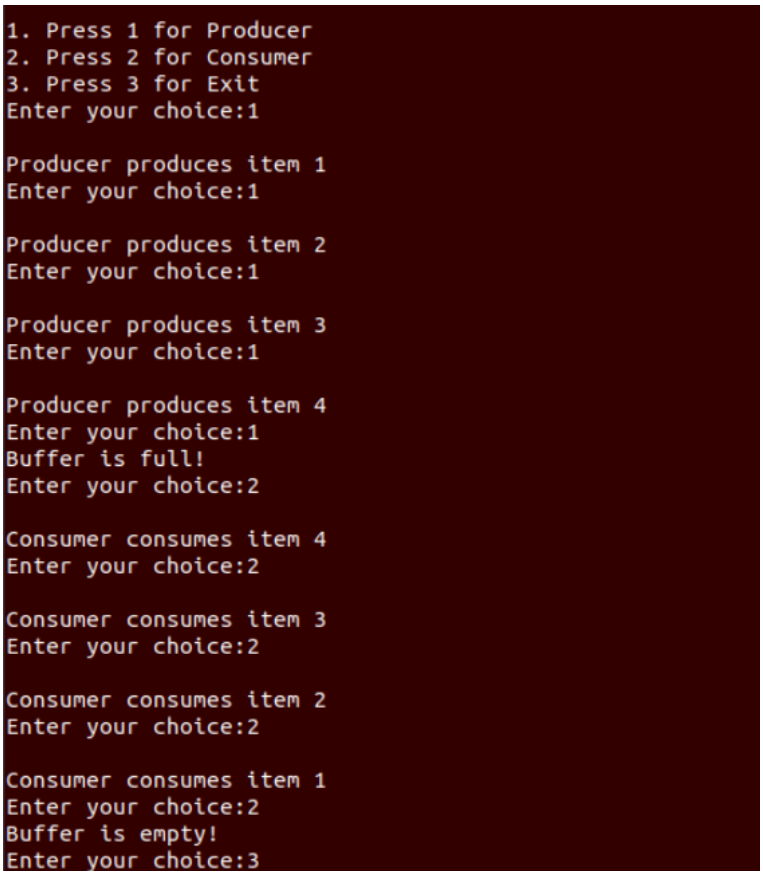
Implement a program for solving Producer Consumer problem using Semaphores

6.2.2 Program Code

```
#include <stdio.h>
#include <stdlib.h>
int mutex = 1;
int full = 0;
int empty = 5, x = 0;
void producer()
{
--mutex;
++full;
--empty;
x++;
printf("\nProducer produces item %d",x);
++mutex;
}
void consumer()
{
--mutex;
--full;
++empty;
printf("\nConsumer consumes item %d",x);
x--;
++mutex;
}
// Driver Code
int main()
{
int n, i;
printf("\n1. Press 1 for Producer"
"\n2. Press 2 for Consumer"
"\n3. Press 3 for Exit");
for (i = 1; i > 0; i++)
{
printf("\nEnter your choice:");
scanf("%d", &n);
switch (n)
{
case 1:
if ((mutex == 1)&& (empty != 0)) {
producer();
}
else {
printf("Buffer is full!");
}
break;
```

```
case 2:
if ((mutex == 1)&& (full != 0)) {
consumer();
}
else {
printf("Buffer is empty!");
}
break;
case 3:
exit(0);
break;
}
}
}
```

6.2.3 Sample Input and Output



6.3 Reader and Writer Problem using Semaphore

6.3.1 Aim

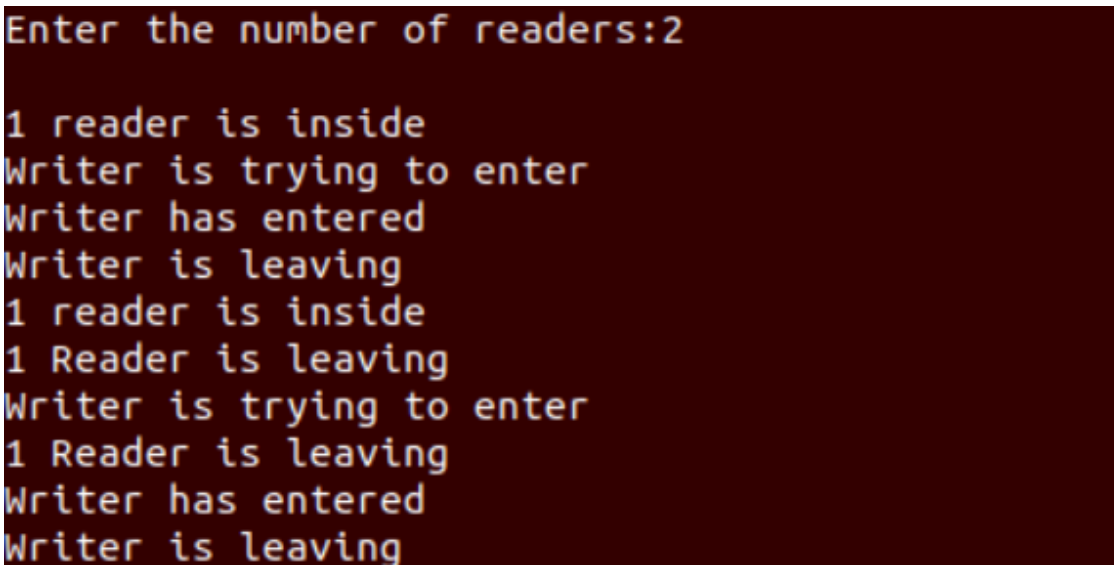
Implement a program for solving Reader Writer problem using Semaphores.

6.3.2 Program Code

```
#include<semaphore.h>
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
sem_t x,y;
pthread_t tid;
pthread_t writerthreads[100],readerthreads[100];
int readercount = 0;
void **reader(void** param)
{
sem_wait(&x);
readercount++;
if(readercount==1)
sem_wait(&y);
sem_post(&x);
printf("%d reader is inside\n",readercount);
usleep(3);
sem_wait(&x);
readercount--;
if(readercount==0)
{
sem_post(&y);
}
sem_post(&x);
printf("%d Reader is leaving\n",readercount+1);
return NULL;
}
void **writer(void** param)
{
printf("Writer is trying to enter\n");
sem_wait(&y);
printf("Writer has entered\n");
sem_post(&y);
printf("Writer is leaving\n");
return NULL;
}
int main()
{
int n2,i;
printf("Enter the number of readers:");
scanf("%d",&n2);
printf("\n");
int n1[n2];
```

```
sem_init(&x,0,1);
sem_init(&y,0,1);
for(i=0;i<n2;i++)
{
pthread_create(&writerthreads[i],NULL,reader,NULL);
pthread_create(&readerthreads[i],NULL,writer,NULL);
}
for(i=0;i<n2;i++)
{
pthread_join(writerthreads[i],NULL);
pthread_join(readerthreads[i],NULL);
}
}
```

6.3.3 Sample Input and Output



6.4 Result

Executed Dining Philosopher's, Reader Writer's and Producer Consumer Problem using semaphore successfully.

7 CPU Scheduling Algorithms

7.1 Aim

Implement the following CPU scheduling algorithms.Using these algorithm find turnaround time and waiting time. For all the programs, arrival time for the processes should be considered and a Gantt chart should be displayed as output. Output should be verified with atleast 5 processes. a)Round Robin b) SJF c) FCFS d) Priority

7.2 First Come First Served(FCFS) Scheduling

7.2.1 Algorithm

1.Start
2.Read the number of processes and their arrival time and burst time.
3.Sort the processes based on their arrival time in ascending order.
4.Initialize the completion time of the first process to the arrival time of the first process.
5.For each process in the sorted list, do the following:
5.1.If it is the first process, calculate its waiting time as 0, turn around time as the burst time.
5.2.If it is not the first process, check if the current time is less than the arrival time of the current process. If it is, print an IDLE state in the Gantt chart until the arrival time of the current process.
5.3.Print the Gantt chart representation of the current process and update the completion time to the sum of the current time and the burst time of the current process.
5.4.Calculate the waiting time of the current process as the difference between its completion time and its arrival time ,and the turnaround time as the sum of its waiting time and its burst time .
6.Print the completion time of the last process.
7.Print the process table and the average waiting time and turnaround time.
8.Stop

7.2.2 Program Code

```
#include <stdio.h>
typedef struct process
{
int pid, arrival, burst, wait, turn;
} process;
void sort(process *p, int n)
{
for (int i = 0; i < n - 1; i++)
{
int min = i;
for (int j = i + 1; j < n; j++)
{
if (p[min].arrival > p[j].arrival)
min = j;
```

```
}
if (i != min)
{
process temp = p[i];
p[i] = p[min];
p[min] = temp;
}
}
}
void fcfs(process *p, int n)
{
p[0].wait = 0;
p[0].turn=p[0].burst;
int sum=p[0].arrival;
for (int i = 1; i < n; i++)
{
sum+=p[i-1].burst;
p[i].wait = sum-p[i].arrival;
if(p[i].wait<0)
p[i].wait=0;
p[i].turn = p[i].burst + p[i].wait;
}
}
void gantt_chart(process *p,int n)
{
printf("\n\n");
for (int i = 0; i < p[n-1].turn - 1; i++)
{
printf(" ");
}
printf("GANTT CHART\n\n ");
int i, j;
for (i = 0; i < n; i++)
{
for (j = 0; j < p[i].burst; j++)
printf("--");
printf(" ");
}
printf("\n|");
for (i = 0; i < n; i++)
{
for (j = 0; j < p[i].burst - 1; j++)
printf(" ");
printf("P%d", i + 1);
for (j = 0; j < p[i].burst - 1; j++)
printf(" ");
printf("|");
}
printf("\n ");
for (i = 0; i < n; i++)
{
for (j = 0; j < p[i].burst; j++)
```

```
printf("--");
printf(" ");
}
printf("\n%d",p[0].arrival);
for (i = 0; i < n; i++)
{
for (j = 0; j < p[i].burst; j++)
printf(" ");
if (p[i].turn+p[i].arrival > 9)
printf("\b");
printf("%d", p[i].turn+p[i].arrival);
}
printf("\n");
}
void average(process *p,int n)
{
float sumw = 0, sumt = 0;
for (int i = 0; i < n; i++)
{
sumw += p[i].wait;
sumt += p[i].turn;
}
printf("\n\nAverage Waiting Time = %0.3f", sumw / n);
printf("\nAverage Turnaround Time = %0.3f\n", sumt / n);
}
void main()
{
int n ;
printf("Enter the number of processes: ");
scanf("%d",&n);
process p[10];
printf("Enter the arrival times of Processes:");
for (int i = 0; i < n; i++)
scanf("%d", &p[i].arrival);
printf("Enter the burst times of Processes:");
for (int i = 0; i < n; i++)
{
scanf("%d", &p[i].burst);
p[i].pid = i + 1;
}
sort(p, n);
fcfs(p, n);
printf("\n---------------------\n");
printf("Processes Arrival time Burst time Waiting
time Turnaround time\n");
printf("-----------------------\n");
for (int i = 0; i < n; i++)
{
printf(" P%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
p[i].pid,p[i].arrival, p[i].burst, p[i].wait,
p[i].turn);
}
```

```
printf("----------------------\n");
gantt_chart(p,n);
average(p, n);
}
```

7.2.3 Sample Input and Output

```
s21b46@administrator-rusa:~/OSL/exp7$ ./a.out
Enter the number of processes: 3
Enter the arrival times of Processes:1 2 3
Enter the burst times of Processes:4 9 3

---------------------------------------------------------------------
Processes       Arrival time    Burst time      Waiting time    Turnaround time
---------------------------------------------------------------------
P1              1               4               0               4
P2              2               9               3               12
P3              3               3               11              14



            GANTT CHAR
 -------- ------------------ ------
|   P1   |        P2        |  P3  |
 -------- ------------------ ------
1        5                 14     17

Average Waiting Time = 4.667
Average Turnaround Time = 10.000
```

7.3 Round Robin Scheduling

7.3.1 Algorithm

1.Start
2.Read the number of processes and their arrival time and burst time.
3.Read the time quantum.
4.Sort the processes based on their arrival time in increasing order.
5.Initialize the current time to 0,If the arrival time of the first process is not 0,set the current time to the arrival time of the first process.
6.Enqueue the first process in the ready queue.
7.While the queue is not empty, repeat the following steps:
7.1. Dequeue the front process from the queue.
7.2. For each process in the array of processes, if the arrival time of the process is equal to the arrival time of the current process and the process is not the current process, enqueue the process.
7.3. Determine the execution time of the current process based on whether its remaining time is less than or equal to the time quantum.
7.4. Reduce the remaining time of the current process by the execution time .
7.5. Update the current time by adding the execution time.
7.6. Print the Gantt chart for the current process .
7.7. For each process in the array of processes, if the arrival time of the process is greater than the previous current time and less than or equal to the current time and the process is not the current process, enqueue the process.
7.8. If the remaining time of the current process is greater than 0,enqueue the current process back to the queue and update its turn-around time and waiting time accordingly.
7.9. Otherwise,update the turn-around time and waiting time of the current process based on the current time and the arrival time and burst time of the process.
7.10. If the queue is now empty, enqueue the next process that arrives after the current time to the queue, print "idle" in the Gantt chart, update the current time to the arrival time of the next process, and break out of the loop.
8.Print the process table and the average waiting time and turnaround time.
9.Stop

7.3.2 Program Code

```
#include <stdio.h>
typedef struct process
{
int pid, arrival, burst, wait, turn;
} process;
void sort(process *p, int n)
{
for (int i = 0; i < n - 1; i++)
{
int min = i;
for (int j = i + 1; j < n; j++)
{
if (p[min].arrival > p[j].arrival)
min = j;
```

```
}
if (i != min)
{
process temp = p[i];
p[i] = p[min];
p[min] = temp;
}
}
}
void fcfs(process *p, int n)
{
p[0].wait = 0;
p[0].turn=p[0].burst;
int sum=p[0].arrival;
for (int i = 1; i < n; i++)
{
sum+=p[i-1].burst;
p[i].wait = sum-p[i].arrival;
if(p[i].wait<0)
p[i].wait=0;
p[i].turn = p[i].burst + p[i].wait;
}
}
void gantt_chart(process *p,int n)
{
printf("\n\n");
for (int i = 0; i < p[n-1].turn - 1; i++)
{
printf(" ");
}
printf("GANTT CHART\n\n ");
int i, j;
for (i = 0; i < n; i++)
{
for (j = 0; j < p[i].burst; j++)
printf("--");
printf(" ");
}
printf("\n|");
for (i = 0; i < n; i++)
{
for (j = 0; j < p[i].burst - 1; j++)
printf(" ");
printf("P%d", i + 1);
for (j = 0; j < p[i].burst - 1; j++)
printf(" ");
printf("|");
}
printf("\n ");
for (i = 0; i < n; i++)
{
for (j = 0; j < p[i].burst; j++)
```

```
printf("--");
printf(" ");
}
printf("\n%d",p[0].arrival);
for (i = 0; i < n; i++)
{
for (j = 0; j < p[i].burst; j++)
printf(" ");
if (p[i].turn+p[i].arrival > 9)
printf("\b");
printf("%d", p[i].turn+p[i].arrival);
}
printf("\n");
}
void average(process *p,int n)
{
float sumw = 0, sumt = 0;
for (int i = 0; i < n; i++)
{
sumw += p[i].wait;
sumt += p[i].turn;
}
printf("\n\nAverage Waiting Time = %0.3f", sumw / n);
printf("\nAverage Turnaround Time = %0.3f\n", sumt / n);
}
void main()
{
int n ;
printf("Enter the number of processes: ");
scanf("%d",&n);
process p[10];
printf("Enter the arrival times of Processes:");
for (int i = 0; i < n; i++)
scanf("%d", &p[i].arrival);
printf("Enter the burst times of Processes:");
for (int i = 0; i < n; i++)
{
scanf("%d", &p[i].burst);
p[i].pid = i + 1;
}
sort(p, n);
fcfs(p, n);
printf("\n---------------------\n");
printf("Processes Arrival time Burst time Waiting
time Turnaround time\n");
printf("-----------------------\n");
for (int i = 0; i < n; i++)
{
printf(" P%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
p[i].pid,p[i].arrival, p[i].burst, p[i].wait,
p[i].turn);
}
```

```
printf("----------------------\n");
gantt_chart(p,n);
average(p, n);
}
```

7.3.3 Sample Input and Output

```
s21b46@administrator-rusa:~/OSL/exp7$ ./a.out
Enter the number of processes: 5
Enter the time quantum : 3
Enter the arrival times of Processes:0 5 1 6 8
Enter the burst times of Processes:8 2 7 3 5


ROUND ROBIN SCHEDULING
------------------------------------------------------------------------
Processes  Arrival time  Burst time  Waiting Time  Turnaround time
------------------------------------------------------------------------
P1              0              8              14              22
P3              1              7              15              22
P2              5              2              4               6
P4              6              3              5               8
P5              8              5              12              17
------------------------------------------------------------------------


                      GANTT CHART

 ------ ------ ------ ---- ------ ------ ------ ---- -- ----
|  P1  |  P3  |  P1  | P2 |  P4  |  P3  |  P5  | P1 |P3| P5 |
 ------ ------ ------ ---- ------ ------ ------ ---- -- ----
0      3      6      9   11     14     17     20   22 23   25


Average Waiting Time = 10.000
Average Turnaround Time = 15.000
```

7.4 Shortest Job First(SJF)

7.4.1 Algorithm

1.Start
2.Sort the processes based on their burst time in ascending order.
3.Initialize the variables completed, prev, curr, and ct to 0, -1, -1, and 0, respectively.
4.While the number of completed processes is less than the total number of processes do the following:
4.1. Set curr to -1.
4.2. For each process in the array of processes, if the process has arrived, has remaining time greater than 0, and either curr is -1 or the rt of the current process is less than the rt of the process pointed by curr, set curr to the index of the current process.
4.3. If curr is not -1 (i.e., a process has been selected for execution), do the following:
4.3.1. If the previously executed process is not the same as the currently selected process, print the start time and the process ID of the current process in the Gantt chart.
4.3.2. Reduce the remaining time of the current process by 1.
4.3.3. Increment the current time by 1.
4.3.4. If the remaining time of the current process is 0, do the following:
a. Increment the variable completed.
b. Compute the turn-around time of the current process as the difference between the current time and the arrival time of the process.
c. Compute the waiting time of the current process as the difference between the tat and the burst time of the process.
4.3.5. Set prev to curr.
4.4. Otherwise do the following:
4.4.1. If the previously executed process (prev) is not -1 (i.e., there was a previous process), print the start time and "IDLE" in the Gantt chart.
4.4.2. Set prev to -1.
4.4.3. Increment the current time by 1.
5.Print the process table and the average waiting time and turnaround time.
6.Stop

7.4.2 Program Code

```
#include <stdio.h>
typedef struct process
{
int pid, arrival, burst, wait, turn;
} process;
void sort_arrival_time(process *p, int n)
{
for (int i = 0; i < n - 1; i++)
{
int min = i;
for (int j = i + 1; j < n; j++)
{
if (p[min].arrival > p[j].arrival)
```

```
{
min = j;
}
}
if (i != min)
{
process temp = p[i];
p[i] = p[min];
p[min] = temp;
}
}
}
void sjfs(process *p, int n)
{
int previous = p[0].arrival;
for (int i = 0; i < n; i++)
{
int min = p[i].burst;
int min_index=i;
int flag = 0;
for ( int j = i +1; j < n ; j++)
{
if(p[j].arrival>previous)
break ;
if(p[j].burst<min)
{
min=p[j].burst;
min_index=j;
flag=1;
}
}
if (flag == 1)
{
process temp = p[i];
p[i] = p[min_index];
p[min_index] = temp;
}
p[i].wait = previous - p[i].arrival;
if (p[i].wait < 0)
p[i].wait = 0;
p[i].turn = p[i].burst + p[i].wait;
previous=p[i].arrival+p[i].turn;
}
}
void gantt_chart(process *p, int n)
{
printf("\n\n");
for (int i = 0; i < p[n - 1].turn - 1; i++)
{
printf(" ");
}
printf("GANTT CHART\n\n ");
```

```
int i, j;
for (i = 0; i < n; i++)
{
for (j = 0; j < p[i].burst; j++)
printf("--");
printf(" ");
}
printf("\n|");
for (i = 0; i < n; i++)
{
for (j = 0; j < p[i].burst - 1; j++)
printf(" ");
// printf("P%d", i + 1);
printf("P%d", p[i].pid);
for (j = 0; j < p[i].burst - 1; j++)
printf(" ");
printf("|");
}
printf("\n ");
for (i = 0; i < n; i++)
{
for (j = 0; j < p[i].burst; j++)
printf("--");
printf(" ");
}
printf("\n%d", p[0].arrival);
for (i = 0; i < n; i++)
{
for (j = 0; j < p[i].burst; j++)
printf(" ");
if (p[i].turn + p[i].arrival > 9)
printf("\b");
printf("%d", p[i].turn + p[i].arrival);
}
printf("\n");
}
void average(process *p, int n)
{
float sumw = 0, sumt = 0;
for (int i = 0; i < n; i++)
{
sumw += p[i].wait;
sumt += p[i].turn;
}
printf("\n\nAverage Waiting Time = %0.3f", sumw / n);
printf("\nAverage Turnaround Time = %0.3f\n", sumt / n);
}
void main()
{
int n;
printf("Enter the number of processes: ");
scanf("%d", &n);
```

```
process p[10];
printf("Enter the arrival times of Processes:");
for (int i = 0; i < n; i++)
scanf("%d", &p[i].arrival);
printf("Enter the burst times of Processes:");
for (int i = 0; i < n; i++)
{
scanf("%d", &p[i].burst);
p[i].pid = i + 1;
}
sort_arrival_time(p, n);
sjfs(p, n);
printf("\n\nSJF - non preemptive\n");
printf("\n-----------------------------\n");
printf("processes Arrival time Burst time Waiting time Turnaround time\n");
printf("-----------------------------\n");
for (int i = 0; i < n; i++)
{
printf("P%d\t\t%d\t\t%d\t\t%d\t\t%d\n", p[i].pid,p[i].arrival, p[i].burst,
p[i].wait, p[i].turn);
}
printf("-----------------------------\n");
gantt_chart(p, n);
average(p, n);
}
```

7.4.3 Sample Input and Output

```
s21b46@administrator-rusa:~/OSL/exp7$ ./a.out
Enter the number of processes: 4
Enter the arrival times of Processes:0 1 2 3 4
Enter the burst times of Processes:3 9 5 1 7

---------------------------------------------------------------------
Processes  Arrival time  Burst time  Waiting time  Turnaround time
---------------------------------------------------------------------
P1              0               4               0               4
P2              1               3               3               6
P4              3               5               4               9
P3              2               9               10              19
---------------------------------------------------------------------


                GANTT CHART

 -------- ------ ---------- ------------------
|   P1   |  P2  |    P3    |        P4        |
 -------- ------ ---------- ------------------
0        4      7         12                21


Average Waiting Time = 4.250
Average Turnaround Time = 9.500
```

7.5 Shortest Remaining Time First (SRTF) Scheduling

7.5.1 Algorithm

1. Start by initializing an empty ready queue to hold the processes.
2. Create a variable called "current_process" to track the currently running process.
3. Read the input processes along with their burst times and arrival times.
4. Sort the processes based on their arrival times in ascending order.
5. Set the current time to the arrival time of the first process.
6. Add the first process to the ready queue.
7. While there are processes in the ready queue:
7.1. Check if the "current_process" is null or the remaining burst time of the "current_process" is greater than the burst time of the next process in the ready queue.
7.1.1. If true, update the "current_process" to the process with the shortest burst time from the ready queue.
7.2. Update the remaining burst time of the "current_process" by subtracting the current time from the arrival time of the next process.
7.3. Update the current time to the arrival time of the next process.
7.4. If the remaining burst time of the "current_process" is 0, remove it from the ready queue and mark it as completed.
7.5. If the next process has arrived, add it to the ready queue.
7.6. Sort the ready queue based on the remaining burst times of the processes.
7.7. Repeat the above steps until all processes are completed.
8. Calculate the waiting time and turnaround time for each process.
9. Calculate the average waiting time and average turnaround time by summing up the waiting times and turnaround times respectively and dividing them by the total number of processes.
10. Display the scheduling results, including the average waiting time and average turnaround time.

7.5.2 Program Code

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int f=-1,r=-1;
int m=0;
int *pid;
int *time;
int idx=-1;
#define max 30
typedef struct process{
int pid,at,bt,brt,priority,ct,tat,wt,temp;
}process;
struct gantt{
int pid,starttime;
} s[30];
void insertGantt(int id,int start){
idx++;
s[idx].pid=id;
s[idx].starttime=start;
```

```
}
void initialize(int completed[],int visited[],int present[],int n){
for(int i=0;i<n;i++){
completed[i]=0;
visited[i]=0;
present[i]=0;
}
}
void initialize_gantt(){
for(int i=0;i<30;i++){
s[i].pid=0;
s[i].starttime=0;
}
}
void arrivalSort(process *p,int n){
for(int i=0;i<n-1;i++){
for(int j=0;j<n-i-1;j++){
if(p[j].at>p[j+1].at){
process temp=p[j];
p[j]=p[j+1];
p[j+1]=temp;
}
}
}
}
void computeTurnWait(process *p,int n){
for(int i=0;i<n;i++){
p[i].tat=p[i].ct-p[i].at;
p[i].wt=p[i].tat-p[i].bt;
}
}
void srtf(process *p,int n,int visited[],int completed[]){
int sumbt=0,i,minbrt,k=0;
for(i=0;i<n;i++){
sumbt+=p[i].bt;
}
int t=p[0].at;
insertGantt(p[0].pid,t);
while(t<sumbt){
for(i=0;i<n;i++){
if(p[i].at<=t && completed[i]==0){
visited[i]=1;
}
}
minbrt=999;
for(i=0;i<n;i++){
if(visited[i] && completed[i]==0){
if(p[i].brt<minbrt){
minbrt=p[i].brt;
k=i;
}
}
```

```
}
p[k].brt=p[k].brt-1;
if(p[k].brt==0){
completed[k]=1;
visited[k]=0;
p[k].ct=t+1;
}
insertGantt(k+1,t);
t++;
}
insertGantt(k+1,t);
computeTurnWait(p,n);
}
void gantt(process *p, int n,int size)
{
printf("\n\n");
for(int i=0;i<(time[size-1]*2+size-11)/2;i++)
printf(" ");
printf("GANTT CHART\n\n ");
int i,j;
for (i = 0; i < size; i++){
for (j = 0; j < time[i]-time[i-1]; j++)
printf("--");
printf(" ");
}
printf("\n|");
for (i = 0; i < size; i++){
for (j = 0; j < time[i]-time[i-1] - 1; j++)
printf(" ");
printf("P%d", pid[i]);
for (j = 0; j < time[i]-time[i-1] - 1; j++)
printf(" ");
printf("|");
}
printf("\n ");
for (i = 0; i < size; i++){
for (j = 0; j < time[i]-time[i-1] ; j++)
printf("--");
printf(" ");
}
printf("\n%d", p[0].at);
for (i = 0; i < size; i++){
for (j = 0; j < time[i]-time[i-1] ; j++)
printf(" ");
if (time[i] > 9)
printf("\b");
printf("%d", time[i]);
}
printf("\n");
}
void computeAverage(process *p,int n){
int sumtat=0;
```

```
int sumwt=0;
for(int i=0;i<n;i++){
sumtat+=p[i].tat;
sumwt+=p[i].wt;
}
printf("Average TurnAround Time = %.2f\n",(double)sumtat/n);
printf("Average Waiting Time = %.2f\n",(double)sumwt/n);
}
void display(process *p,int n){
printf("Process no Arrival Time Burst Time Completion Time
TurnAround Time Waiting Time\n\n");
for(int i=0;i<n;i++){
printf(" %d %2d %2d %2d
%2d %2d\n",i+1,p[i].at,p[i].bt,p[i].ct,p[i].tat,p[i].wt);
}
printf("\n");
}
void gantt_chart(){
int i=0;
printf("-----");
while(s[i+1].pid!=0){
if(s[i].pid!=s[i+1].pid){
printf("------");
}
else
printf("-");
i++;
}
printf("-\n");
i=0;
printf("| P%d ",s[i].pid);
while(s[i+1].pid!=0){
if(s[i].pid!=s[i+1].pid){
printf(" | P%d ",s[i+1].pid);
}
else
printf(" ");
i++;
}
printf("|\n");
i=0;
printf("-----");
while(s[i+1].pid!=0){
if(s[i].pid!=s[i+1].pid){
printf("------");
}
else
printf("-");
i++;
}
printf("-\n");
i=0;
```

```
printf("%-2d ",s[i].starttime);
while(s[i+1].pid!=0){
if(s[i].pid!=s[i+1].pid){
printf(" %-2d ",s[i+1].starttime);
}
else
printf(" ");
i++;
}
printf("%-2d\n",s[i].starttime);
}
void main(){
printf("Enter the number of processes : ");
int n;
scanf("%d",&n);
int quantum,size=0;
process *p=(process *)malloc(n*sizeof(process));
int *completed=(int *)malloc(n*sizeof(int));
int *visited=(int *)malloc(n*sizeof(int));
int *present=(int *)malloc(n*sizeof(int));
printf("Enter the arrival time of the processes : ");
for(int i=0;i<n;i++){
scanf("%d",&p[i].at);
}
printf("Enter the burst time of the processes : ");
for(int i=0;i<n;i++){
scanf("%d",&p[i].bt);
p[i].brt=p[i].bt;
p[i].pid=i+1;
p[i].temp = p[i].at;
float f=p[i].bt/(float)quantum;
size+=ceil(f);
}
int choice;
int timeslice;
pid = (int *)malloc(sizeof(int)*size);
time = (int *)malloc(sizeof(int)*size);
initialize(completed,visited,present,n);
initialize_gantt();
arrivalSort(p,n);
srtf(p,n,visited,completed);
display(p,n);
gantt_chart();
printf("\n");
computeAverage(p,n);
}
```

7.5.3 Sample Input and Output

```
Enter the number of processes : 5
Enter the arrival time of the processes : 0 1 2 3 4
Enter the burst time of the processes : 3 5 2 1 3
Process no Arrival Time Burst Time Completion Time TurnAround Time Waiting Time

 1  0  3  3  3  0
 2  1  5 14 13  8
 3  2  2  6  4  2
 4  3  1  4  1  0
 5  4  3  9  5  2

----------------------------------------
| P1     | P4  | P3   | P5    | P2      |
----------------------------------------
0      3   4    6     9       14

Average TurnAround Time = 5.20
Average Waiting Time = 2.40
```

7.6 Priority Scheduling - Non Preemptive

7.6.1 Algorithm

1.Start
2.Input the number of processes and their details including arrival time,
burst time, and priority.
3.Sort the processes in ascending order based on their arrival time.
4.Initialize the variables completed, ct, curr, and prev.
5.While the number of completed processes is less than n, do the following:
5.1.Set curr to -1.
5.2.Loop through all the processes to find the process with the highest
priority .If no process is found, break the loop.
5.3.If a process is found, update the Gantt chart by printing the current
time and the process ID. If the current process is different from the
previous process, update prev to curr.
5.4.Decrement the remaining time of the current process by 1, increment
the current time by 1, and check if the current process has completed execution
5.5.If the current process has completed execution, update its turnaround
time and waiting time, increment completed by 1, and set prev to -1.
5.6.If the current process has not completed execution, update prev to curr.
5.7.If no process can be executed at the current time, update the Gantt
chart by printing the current time and "IDLE", and increment the current time by 1.
6.Print the completion time of the last process.
7.Print the process table and the average waiting time and turnaround time.

7.6.2 Program Code

```
#include <stdio.h>
typedef struct process
{
int pid, arrival, burst, wait, turn,priority;
} process;
void sort_arrival_time(process *p, int n)
{
```

```
for (int i = 0; i < n - 1; i++)
{
int min = i;
for (int j = i + 1; j < n; j++)
{
if (p[min].arrival > p[j].arrival)
{
min = j;
}
}
if (i != min)
{
process temp = p[i];
p[i] = p[min];
p[min] = temp;
}
}
}
void priority(process *p, int n)
{
int previous = p[0].arrival;
for (int i = 0; i < n; i++)
{
int min = p[i].priority;
int min_index=i;
int flag = 0;
for (int j=i+1;j<n;j++)
{
if(p[j].arrival>previous)
break;
if (p[j].priority < min)
{
min = p[j].priority;
min_index = j;
flag = 1;
}
}
if (flag == 1)
{
process temp = p[i];
p[i] = p[min_index];
p[min_index] = temp;
}
p[i].wait = previous - p[i].arrival;
if (p[i].wait < 0)
p[i].wait = 0;
p[i].turn = p[i].burst + p[i].wait;
previous=p[i].arrival+p[i].turn;
}
}
void gantt_chart(process *p, int n)
{
```

```
printf("\n\n");
for (int i = 0; i < p[n - 1].turn - 1; i++)
printf(" ");
printf("GANTT CHART\n\n ");
int i, j;
for (i = 0; i < n; i++)
{
for (j = 0; j < p[i].burst; j++)
printf("--");
printf(" ");
}
printf("\n|");
for (i = 0; i < n; i++)
{
for (j = 0; j < p[i].burst - 1; j++)
printf(" ");
printf("P%d", p[i].pid);
for (j = 0; j < p[i].burst - 1; j++)
printf(" ");
printf("|");
}
printf("\n ");
for (i = 0; i < n; i++)
{
for (j = 0; j < p[i].burst; j++)
printf("--");
printf(" ");
}
printf("\n%d", p[0].arrival);
for (i = 0; i < n; i++)
{
for (j = 0; j < p[i].burst; j++)
printf(" ");
if (p[i].turn + p[i].arrival > 9)
printf("\b");
printf("%d", p[i].turn + p[i].arrival);
}
printf("\n");
}
void average(process *p, int n)
{
float sumw = 0, sumt = 0;
for (int i = 0; i < n; i++)
{
sumw += p[i].wait;
sumt += p[i].turn;
}
printf("\n\nAverage Waiting Time = %0.3f", sumw / n);
printf("\nAverage Turnaround Time = %0.3f\n", sumt / n);
}
void main()
{
```

```
int n;
printf("Enter the number of processes: ");
scanf("%d", &n);
process p[10];
printf("Enter the arrival times of Processes:");
for (int i = 0; i < n; i++)
scanf("%d", &p[i].arrival);
printf("Enter the burst times of Processes:");
for (int i = 0; i < n; i++)
{
scanf("%d", &p[i].burst);
p[i].pid = i + 1;
}
printf("Enter the priorities of Processes:");
for (int i = 0; i < n; i++)
{
scanf("%d", &p[i].priority);
}
sort_arrival_time(p, n);
priority(p, n);
printf("\n\nPRIORITY SCHEDULING - non preemptive");
printf("\n----------\n");
printf("processes Arrival time Burst time Priority Waiting time Turnaround time\n");
printf("----------\n");
for (int i = 0; i < n; i++)
{
printf("P%d\t\t%d\t\t%d\t%d\t\t%d\t\t%d\n",p[i].pid, p[i].arrival,
p[i].burst,p[i].priority,p[i].wait, p[i].turn);
}
printf("----------\n");
gantt_chart(p, n);
average(p, n);
}
```

7.6.3 Sample Input and Output

```
s21b46@administrator-rusa:~/OSL/exp7$ ./a.out
Enter the number of processes: 3
Enter the arrival times of Processes:1 2 3
Enter the burst times of Processes:8 2 6
Enter the priorities of Processes:4 2 3


PRIORITY SCHEDULING
-------------------------------------------------------------------------------------
Processes  Arrival time  Burst time  Priority  Waiting time  Turnaround time
-------------------------------------------------------------------------------------
P1              1              8       4              0              8
P2              2              2       2              7              9
P3              3              6       3              8              14
-------------------------------------------------------------------------------------


            GANTT CHART

 ---------------- ---- ------------
|       P1       | P2 |     P3     |
 ---------------- ---- ------------
1               9   11           17


Average Waiting Time = 5.000
Average Turnaround Time = 10.333
```

7.7 Priority Scheduling - Preemptive

7.7.1 Algorithm

1.Start
2.Input the number of processes and their details including arrival time,
burst time, and priority.
3.Sort the processes in ascending order based on their arrival time.
4.Initialize the variables completed, ct, curr, and prev.
5.While the number of completed processes is less than n, do the following:
5.1.Set curr to -1.
5.2.Loop through all the processes to find the process with the highest
priority .If no process is found, break the loop.
5.3.If a process is found, update the Gantt chart by printing the current
time and the process ID. If the current process is different from the
previous process, update prev to curr.
5.4.Decrement the remaining time of the current process by 1, increment
the current time by 1, and check if the current process has completed execution
5.5.If the current process has completed execution, update its turnaround
time and waiting time, increment completed by 1, and set prev to -1.
5.6.If the current process has not completed execution, update prev to curr.
5.7.If no process can be executed at the current time, update the Gantt
chart by printing the current time and "IDLE", and increment the current time by 1.
6.Print the completion time of the last process.
7.Print the process table and the average waiting time and turnaround time.

7.7.2 Program Code

```
#include<stdio.h>
#define MIN -9999;
struct proc
{
int no,at,bt,rt,ct,wt,tat,pri,temp;
};
struct proc read(int i)
{
struct proc p;
printf("\nProcess No: %d\n",i);
p.no=i;
printf("Enter Arrival Time: ");
scanf("%d",&p.at);
printf("Enter Burst Time: ");
scanf("%d",&p.bt);
p.rt=p.bt;
printf("Enter Priority: ");
scanf("%d",&p.pri);
p.temp=p.pri;
return p;
}
void main()
{
int i,n,c,remaining,max_val,max_index;
```

```
struct proc p[10],temp;
float avgtat=0,avgwt=0;
printf("<--Highest Priority First Scheduling Algorithm (Preemptive)-->\n");
printf("Enter Number of Processes: ");
scanf("%d",&n);
for(int i=0;i<n;i++)
p[i]=read(i+1);
remaining=n;
for(int i=0;i<n-1;i++)
for(int j=0;j<n-i-1;j++)
if(p[j].at>p[j+1].at)
{
temp=p[j];
p[j]=p[j+1];
p[j+1]=temp;
}
max_val=p[0].temp,max_index=0;
for(int j=0;j<n&&p[j].at<=p[0].at;j++)
if(p[j].temp>max_val)
max_val=p[j].temp,max_index=j;
i=max_index;
c=p[i].ct=p[i].at+1;
p[i].rt--;
if(p[i].rt==0)
{
p[i].temp=MIN;
remaining--;
}
while(remaining>0)
{
max_val=p[0].temp,max_index=0;
for(int j=0;j<n&&p[j].at<=c;j++)
if(p[j].temp>max_val)
max_val=p[j].temp,max_index=j;
i=max_index;
p[i].ct=c=c+1;
p[i].rt--;
if(p[i].rt==0)
{
p[i].temp=MIN;
remaining--;
}
}
printf("\nProcessNo\tAT\tBT\tPri\tCT\tTAT\tWT\n");
for(int i=0;i<n;i++)
{
p[i].tat=p[i].ct-p[i].at;
avgtat+=p[i].tat;
p[i].wt=p[i].tat-p[i].bt;
avgwt+=p[i].wt;
printf("P%d\t\t%d\t%d\t%d\t%d\t%d\t%d\n",
p[i].no,p[i].at,p[i].bt,p[i].pri,p[i].ct,p[i].tat,p[i].wt);
```

```
}
avgtat/=n,avgwt/=n;
printf("\nAverage TurnAroundTime=%f\nAverage WaitingTime=%f\n",avgtat,avgwt);
}
```

7.7.3 Sample Input and Output

```
Enter Number of Processes: 5

Process No: 1
Enter Arrival Time: 0
Enter Burst Time: 15
Enter Priority: 2

Process No: 2
Enter Arrival Time: 2
Enter Burst Time: 3
Enter Priority: 1

Process No: 3
Enter Arrival Time: 5
Enter Burst Time: 5
Enter Priority: 5

Process No: 4
Enter Arrival Time: 6
Enter Burst Time: 8
Enter Priority: 4

Process No: 5
Enter Arrival Time: 7
Enter Burst Time: 12
Enter Priority: 3

ProcessNo       AT      BT      Pri     CT      TAT     WT
P1              0       15      2       40      40      25
P2              2       3       1       43      41      38
P3              5       5       5       10      5       0
P4              6       8       4       18      12      4
P5              7       12      3       30      23      11

Average TurnAroundTime=24.200001
Average WaitingTime=15.600000
```

7.8 Result

Executed CPU Scheduling Algorithms successfully.

8 Page Replacement Algorithms

Aim

Implement the page replacement algorithms a) FIFO b) LRU c) LFU

8.1 First In First Out(FIFO)

8.1.1 Algorithm

```
    1.Start
2.Input the number of frames and the size of the reference string.
3.Input the reference string and store it in the "s" array.
4.Iterate over the reference string "s". For each page "pg" in the reference string:
4.1.If the page is not in the queue ,increment the "fault" counter.
4.2.If the queue is full,remove the page at the front of the queue.
4.3.Enqueue the current page to the end of the queue.
5.Print the number of page faults.
6.Stop
```

8.1.2 Program Code

```
#include <stdio.h>
void main()
{
int i,j,n,m,fnd,pg[100],fr[100],k=0,cnt=0;
printf("ENTER THE NUMBER OF PAGES : ");
scanf("%d",&n);
printf("ENTER THE PAGE NUMBERS: ");
for(i=0;i<n;i++){
scanf("%d",&pg[i]);
}
printf("ENTER THE NUMBER OF FRAMES : ");
scanf("%d",&m);
for(i=0;i<m;i++)
{
fr[i]=-1;
}
printf("\n\tREFERENCE STRING\tPAGE NUMBER\t\t STATUS \n");
for(i=0;i<n;i++)
{
fnd=0;
printf("\t\t%d\t\t",pg[i]);
for(j=0;j<m;j++)
{
if(fr[j]==pg[i])
{
fnd = 1;
}
}
if(fnd == 0)
```

```
{
fr[k] = pg[i];
k = (k+1)%m;
cnt++;
}
for(j=0;j<m;j++)
{
if(fr[j] != -1)
{
printf("%d   ",fr[j]);
}
}
if(fnd==1)
{
printf("\t\tHIT\n");
}
else
{
printf("\t\tMISS\n");
}
}
printf("\nPAGE FAULT : %d\n",cnt);
}
```

8.2 Least Recently Used(LRU)

8.2.1 Algorithm

```
1.Start
2.Input the number of frames and the size of the reference string.
3.Input the reference string and store it in the "s" array.
4.Iterate over the reference string "s". For each page "pg" in the reference string:
4.1.Search for the current page
4.2.If the page is not in the memory:
a.increment the "fault" counter.
b.If the memory is full, remove the least recently used page.
c.Add the current page to the memory.
4.3.If the page is already in memory,place the current page at the top
and remove it from the current position.
5.Print the number of page faults.
6.Stop
```

8.2.2 Program Code

```
#include<stdio.h>
int ref_string[20];
int no_of_frames;
int stack[100],top=-1,size=0;
void push(int item)
{
```

```
top++;
stack[top]=item;
size++;
}
void pop_from_bottom()
{
for(int i=0;i<top;i++)
{
stack[i]=stack[i+1];
}
top--;
size--;
}
int search(int item)
{
for(int i=0;i<=top;i++)
{
if(stack[i]==item)
{
return i;
}
}
return -1;
}
void main()
{
printf("input no. of frames:");
scanf("%d",&no_of_frames);
int n;
printf("\ninput reference string size:");
scanf("%d",&n);
printf("\ninput reference string:");
for(int i=0;i<n;i++)
scanf("%d",&ref_string[i]);
int page_fault=0;
for(int i=0;i<n;i++)
{
int page_no=ref_string[i];
int index=search(page_no);
if(index==-1)
{
page_fault++;
if(size<no_of_frames)
{
push(page_no);
}
else
{
pop_from_bottom();
push(page_no);
}
}
```

```
else
{
int temp=stack[index];
for(int i=index;i<top;i++)
{
stack[i]=stack[i+1];
}
stack[top]=temp;
}
printf("\npages in memory:");
for(int i=0;i<=top;i++)
{
printf("%d ",stack[i]);
}
}
printf("\n\nOUTPUT-");
printf("\nno. of page faults:%d\n",page_fault);
}
```

8.3 Least Frequently Used(LFU)

8.3.1 Algorithm

1.Start
2.Input the number of frames and the size of the reference string.
3.Input the reference string and store it in the "s" array.
4.Iterate over the reference string "s". For each page "pg" in the reference string:
4.1.Search for the current page
4.2.If the page is not in the memory:
a.increment the "fault" counter.
b.If the memory is full, remove the least frequently used page.
c.Add the current page to the memory.
4.3.If the page is already in memory,increment current page's count value
5.Print the number of page faults.
6.Stop

8.3.2 Program Code

```
#include<stdio.h>
int ref_string[20];
int no_of_frames;
typedef struct{
int page_no;
int count; //no. of references
}pages_in_memory;
pages_in_memory pages[20];
int n=-1,size=0;
void add_page(int item)
{
pages_in_memory page;
page.page_no=item;
```

```
page.count=1;
pages[++n]=page;
size++;
}
void remove_page() //removes page with least reference count
{
for(int i=1;i<size;i++)
{
for(int j=0;j<size-i;j++)
{
if(pages[j].count > pages[j+1].count)
{
pages_in_memory temp=pages[j];
pages[j]=pages[j+1];
pages[j+1]=temp;
}
}
}
for(int i=0;i<size-1;i++)
{
pages[i]=pages[i+1];
}
size--;
}
int search(int item)
{
for(int i=0;i<size;i++)
{
if(pages[i].page_no==item)
{
return 1;
}
}
return 0;
}
void main()
{
printf("input no. of frames:");
scanf("%d",&no_of_frames);
int n;
printf("\ninput reference string size:");
scanf("%d",&n);
printf("\ninput reference string:");
for(int i=0;i<n;i++)
scanf("%d",&ref_string[i]);
int page_fault=0;
for(int i=0;i<n;i++)
{
int pno=ref_string[i];
if(search(pno)==0)
{
page_fault++;
```

```
if(size<no_of_frames)
{
add_page(pno);
}
else
{
remove_page();
add_page(pno);
}
}
else
{
for(int i=0;i<size;i++)
7
{
if(pages[i].page_no==pno)
{
pages[i].count++;
break;
}
}
}
}
printf("\nOUTPUT-");
printf("\nno. of page faults:%d\n",page_fault);
}
```

8.4 Sample Input and Output

FIFO

```
s21b46@administrator-rusa:~/OSL/exp8$ ./a.out
ENTER THE NUMBER OF PAGES : 5
ENTER THE PAGE NUMBERS: 1 2 3 2 4
ENTER THE NUMBER OF FRAMES : 3

        REFERENCE STRING        PAGE NUMBER             STATUS
                1               1               MISS
                2               1   2                   MISS
                3               1   2   3               MISS
                2               1   2   3               HIT
                4               4   2   3               MISS

PAGE FAULT : 4
```

LRU

```
s21b46@administrator-rusa:~/OSL/exp8$ cc lfu.c
s21b46@administrator-rusa:~/OSL/exp8$ ./a.out
ENTER THE NUMBER OF PAGES : 5
ENTER THE REFERENCING STRING : 1 3 4 3 2
ENTER THE NUMBER OF FRAMES : 3

REFERENCING PAGE        STATUS          FRAME CONTENT

        1               MISS            1
        3               MISS            1       3
        4               MISS            1       3       4
        3               HIT             1       3       4
        2               MISS            2       3       4

TOTAL PAGE FAULT : 4
```

LRU

```
s21b46@administrator-rusa:~/OSL/exp8$ cc lru.c
s21b46@administrator-rusa:~/OSL/exp8$ ./a.out
ENTER THE NUMBER OF PAGES : 5
ENTER THE REFERENCING STRING : 1 2 3 2 4
ENTER THE NUMBER OF FRAMES : 3

REFERENCING PAGE        STATUS          FRAME CONTENT

        1               MISS            1
        2               MISS            1       2
        3               MISS            1       2       3
        2               HIT             1       4       3
        4               MISS            5       4       3

PAGE FAULT : 4
```

8.5 Result

Executed Page Replacement Algorithms successfully.

9 Banker's Problem

Aim

Implement the Banker's algorithm for deadlock avoidance.

9.1 Algorithm

1.Start
2.Input no of processes and resource types.
3.Input available resources,max matrix,allocation matrix.
4.Initialise need matrix and finish matrix.
5.Loop through all the processes and check if they can be executed:
5.1. Loop through all the processes again to find a process that has not finished yet.
5.2. Check if the remaining resources needed by the process are less than or equal to the available resources.
5.3. If the process can be executed, add it to the safe sequence array and update the available resources array.
5.4. Mark the process as finished.
6.Print the safe sequence
7.Stop

9.2 Program Code

```
#include<stdio.h>
int available[10];
int allocation[10][10];
int max[10][10];
int need[10][10];
void main()
{
int n,m;
printf("input the no.of processes:");
scanf("%d",&n);
printf("input the no.of resource types:");
scanf("%d",&m);
printf("\navailable resources:");
for(int i=0;i<m;i++)
scanf("%d",&available[i]);
printf("\nmax matrix:\n");
for(int i=0;i<n;i++)
{
printf("process %d:",i);
for(int j=0;j<m;j++)
scanf("%d",&max[i][j]);
}
printf("\nallocation matrix:\n");
for(int i=0;i<n;i++)
{
```

```
printf("process %d:",i);
for(int j=0;j<m;j++)
{
scanf("%d",&allocation[i][j]);
need[i][j]=max[i][j]-allocation[i][j];
}
}
int finish[10],safe_sequence[10],z=0;
for (int i=0;i<n;i++)
finish[i]=0;
for(int k=0;k<n;k++)
{
for(int i=0;i<n;i++)
{
if(finish[i]==0)
{
int flag=0;
for (int j=0;j<m;j++)
{
if(need[i][j]>available[j])
{
flag=1;
break;
}
}
if(flag==0)
{
safe_sequence[z++]=i;
for (int y=0;y<m;y++)
available[y]+=allocation[i][y];
finish[i]=1;
}
}
}
}
printf("\nSAFE Sequence is:\n");
for (int i=0;i<n-1;i++)
printf(" P%d ->",safe_sequence[i]);
printf("P%d\n",safe_sequence[n-1]);
}
```

9.3 Sample Input and Output

```
input the no.of processes:3
input the no.of resource types:3

available resources:3 2 1

max matrix:
process 0:2 1 1
process 1:0 1 0
process 2:3 0 0

allocation matrix:
process 0:0 0 1
process 1:0 1 0
process 2:2 0 1

SAFE Sequence is:
 P0 -> P1 ->P2
```

9.4 Result

Executed Banker's Algorithm successfully.

10 Deadlock Detection

Aim

Implement the algorithm for deadlock detection.

10.1 Algorithm

1.Start
2.Input no of processes and resource types.
3.Input available resources,allocation matrix,request matrix.
4.Initialise work matrix.
5.For each process 'i', check if all its allocation entries are zero. If yes, set the corresponding 'finish' entry to 1.
6.Loop through all the processes and check if they can be executed:
6.1. Loop through all the processes again to find a process that has not finished yet.
6.2. Check if the remaining resources needed by the process are less than or equal to the available resources.
6.3. If the process can be executed,mark the process as finished.
7.Print the safe sequence
8.Stop

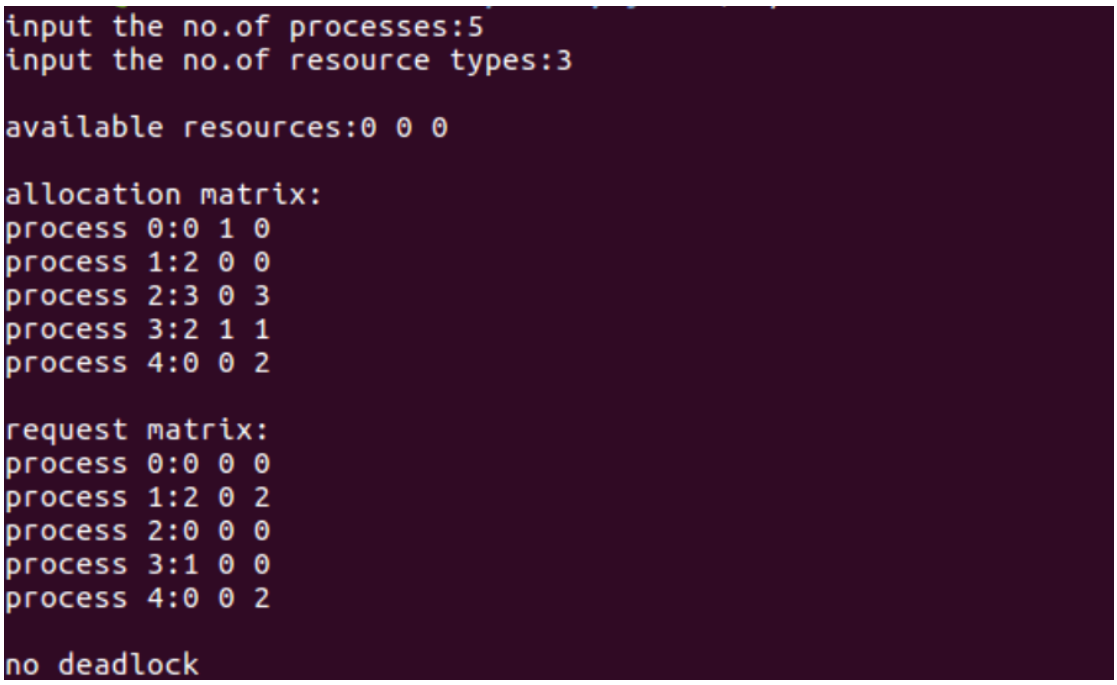
10.2 Program Code

```
#include<stdio.h>
#include<stdlib.h>
int available[10];
int allocation[10][10];
int request[10][10];
void main()
{
int n,m;
printf("input the no.of processes:");
scanf("%d",&n);
printf("input the no.of resource types:");
scanf("%d",&m);
printf("\navailable resources:");
for(int i=0;i<m;i++)
scanf("%d",&available[i]);
printf("\nallocation matrix:\n");
for(int i=0;i<n;i++)
{
printf("process %d:",i);
for(int j=0;j<m;j++)
scanf("%d",&allocation[i][j]);
}
printf("\nrequest matrix:\n");
for(int i=0;i<n;i++)
{
```

```
printf("process %d:",i);
for(int j=0;j<m;j++)
scanf("%d",&request[i][j]);
}
int*work=(int*)malloc(m*sizeof(int));
int*finish=(int*)malloc(n*sizeof(int));
for(int i=0;i<m;i++)
work[i]=available[i];
for(int i=0;i<n;i++)
{
int flag=0;
for(int j=0;j<m;j++)
{
if(allocation[i][j]!=0)
{
flag=1;
break;
}
}
if(flag==0)
finish[i]=1;
else
finish[i]=0;
}
for(int z=0;z<n;z++)
{
for(int i=0;i<n;i++)
{
if(finish[i]==0)
{
int f=0;
for(int j=0;j<m;j++)
{
if(request[i][j]>work[j])
f=1;
}
if(f==0)
{
for(int p=0;p<m;p++)
work[p]+=allocation[i][p];
finish[i]=1;
}
}
}
}
int flag=0;
for(int i=0;i<n;i++)
{
if(finish[i]==0)
{
flag=1;
printf("\ndeadlock detected!\n");
```

```
break;
}
}
if(flag==0)
printf("\nno deadlock\n");
}
```

10.3 Sample Input and Output



10.4 Result

Executed deadlock detection algorithms successfully.

11 Memory Allocation

11.1 Problem

Implement the Memory Allocation Methods for fixed partition. Demonstrate the outputs with at least 2 inputs each. a)First Fit b)Worst Fit c)Best Fit

11.2 Algorithm

11.2.1 First Fit

1.Start
2.Input the number of blocks, processes ,block size and processes size.
3.Initialize array block_allocated to 0 and process-alloc to -1.
4.For each process i in the process_size array:
4.1. For each block j in the block_size array,if the block j has not been allocated and if the block size<=process size , then allocate the block to the process i and break.
5.Print a table with the following columns: Process No, Process size, Allocated block, Allocated blocksize, and Internal fragmentation..
6.Stop

11.2.2 Best Fit

1.Start
2.Input the number of blocks, processes ,block size and processes size.
3.Initialize array block_allocated to 0 and process-alloc to -1.
4.For each process i in the process_size array, do the following:
4.1. For each block j in the block_size array,find the samllest unallocated block having size greater than processess i and then allocate the block to the process i and break.
5.Print a table with the following columns: Process No, Process size, Allocated block, Allocated blocksize, and Internal fragmentation..
6.Stop

11.2.3 Worst Fit

1.Start
2.Input the number of blocks, processes ,block size and processes size.
3.Initialize array block_allocated to 0 and process-alloc to -1.
4.For each process i in the process_size array, do the following:
4.1. For each block j in the block_size array,find the largest unallocated block having size greater than processess i and then allocate the block to the process i and break.
5.Print a table with the following columns: Process No, Process size, Allocated block, Allocated blocksize, and Internal fragmentation..
6.Stop

11.3 Program Code

11.3.1 First Fit

```
#include <stdio.h>
void main(){
int process_size[10],block_size[10],block_allocated[10],process_alloc[10];
int m,n;
printf("Input no. of blocks:");
scanf("%d",&m);
printf("Input no. of processes:");
scanf("%d",&n);
printf("Input block sizes:");
for(int i=0;i<m;i++){
printf("\nblock %d:",i+1);
scanf("%d",&block_size[i]);
}
printf("\ninput process sizes:");
for(int i=0;i<n;i++){
printf("\nprocess %d:",i+1);
scanf("%d",&process_size[i]);
}
for(int i=0;i<m;i++)
block_allocated[i]=0;
for(int i=0;i<n;i++)
process_alloc[i]=-1;
for(int i=0;i<n;i++){
for(int j=0;j<m;j++){
if(block_allocated[j]==0){
if(block_size[j]>=process_size[i]){
block_allocated[j]=1;
process_alloc[i]=j;
break;
}
}
}
}
printf("\nprocesss no.|process size|allocated block|allocated block size|internal
fragmentation\n");
for(int i=0;i<n;i++){
int j=process_alloc[i];
int s=process_size[i];
int frag=block_size[j]-process_size[i];
if(process_alloc[i]!=-1)
printf("%d\t\t%d\t\t%d\t\t%d\t\t %d\n",i+1,s,j+1,block_size[j],frag);
else
printf("%d\t\t%d\t\t--\t\t--\t\t --\n",i+1,s);
}
}
```

11.3.2 Best Fit

```
#include<stdio.h>
int process_size[10],block_size[10],block_allocated[10],process_alloc[10];
void main(){
int m;
printf("input the no. of blocks:");
scanf("%d",&m);
int n;
printf("\ninput the no. of processes:");
scanf("%d",&n);
printf("\ninput block sizes:");
for(int i=0;i<m;i++){
printf("\nblock %d:",i+1);
scanf("%d",&block_size[i]);
}
printf("\ninput process sizes:");
for(int i=0;i<n;i++){
printf("\nprocess %d:",i+1);
scanf("%d",&process_size[i]);
}
for(int i=0;i<m;i++)
block_allocated[i]=0;
for(int i=0;i<n;i++)
process_alloc[i]=-1;
for(int i=0;i<n;i++){
int small;
for(int j=0;j<m;j++){
if(block_allocated[j]==0){
if(block_size[j]>=process_size[i]){
small=j;
break;
}
}
}
for(int j=0;j<m;j++){
if(block_allocated[j]==0){
if(((block_size[j]-process_size[i])>=0)&&(block_size[j]<block_size[small])){
small=j;
}
}
}
process_alloc[i]=small;
block_allocated[small]=1;
}
printf("\nprocesss no.|process size|allocated block|
allocated block size|internal fragmentation\n");
for(int i=0;i<n;i++){
int j=process_alloc[i];
int frag=block_size[j]-process_size[i];
if(process_alloc[i]!=-1)
printf("%d\t\t%d\t\t%d\t\t%d\t\t %d\n",i+1,process_size[i],process_alloc[i]+1,
```

```
 block_size[j],frag);
else
printf("%d\t\t%d\t\t--\t\t--\t\t --\n",i+1,process_size[i]);
}
}
```

11.3.3 Worst Fit

```
#include<stdio.h>
void initialize(int a[],int b[]){
for(int i=0;i<10;i++){
a[i]=0;
b[i]=-1;
}
}
void main(){
int m,n,i,j;
printf("Enter the number of blocks : ");
scanf("%d",&m);
printf("Enter the number of processes : ");
scanf("%d",&n);
int blockSize[10],processSize[10],blockAllocated[10],blockNoAlloc[10];
printf("Enter the block size of each block : ");
for(i=0;i<m;i++){
printf("Block %d - ",i);
scanf("%d",&blockSize[i]);
}
printf("Enter the process size of each process : \n");
for(i=0;i<n;i++){
printf("Process %d - ",i);
scanf("%d",&processSize[i]);
}
initialize(blockAllocated,blockNoAlloc);
int maxdiff,diff;
int k,l;
for(i=0;i<n;i++){
maxdiff=-1;
for(j=0;j<m;j++){
if(blockAllocated[j]==0 && blockSize[j]>=processSize[i]){
diff=blockSize[j]-processSize[i];
if(diff>maxdiff){
maxdiff=diff;
k=i;
l=j;
}
}
}
if(maxdiff==-1){
continue;
}
```

```
blockAllocated[l]=1;
blockNoAlloc[k]=l;
}
printf("\nprocess no.|process size|allocated block|allocated block size|internal fragmentation\n");
for(int i=0;i<n;i++){
int j=blockNoAlloc[i];
int s=processSize[i];
int frag=blockSize[j]-processSize[i];
if(blockNoAlloc[i]!=-1)
printf(" %d\t\t%d\t\t%d\t\t%d\t\t %d\n",i,s,j,blockSize[j],frag);
else
printf(" %d\t\t%d\t\t--\t\t--\t\t --\n",i,s);
}
}
```

11.4 Sample Input and Output

First Fit

```
s21b46@administrator-rusa:~/OSL/exp11$ ./a.out
Input no. of blocks:3
Input no. of processes:3
Input block sizes:
block 1: 30

block 2: 70

block 3: 55

input process sizes:
process 1: 43

process 2: 50

process 3: 13

processs no.|process size|allocated block|allocated block size|internal fragmentation
1               43              2               70              27
2               50              3               55              5
3               13              1               30              17
s21b46@administrator-rusa:~/OSL/exp11$
```

Best Fit

```
s21b46@administrator-rusa:~/OSL/exp11$ ./a.out
input the no. of blocks:3

input the no. of processes:3

input block sizes:
block 1: 50

block 2: 103

block 3: 201

input process sizes:
process 1: 101

process 2: 45

process 3: 150

processs no.|process size|allocated block|allocated block size|internal fragmentation
1               101             2               103             2
2               45              1               50              5
3               150             3               201             51
```

Worst Fit

```
s21b46@administrator-rusa:~/OSL/11.Memory Allocation$ ./a.out
input the no. of blocks:3

input the no. of processes:3

input block sizes:
block 0:10

block 1:40

block 2:70

input process sizes:
process 0:5

process 1:10

process 2:8

processs no.|process size|allocated block|allocated block size|internal fragmentation
0               5               0               10              5
1               10              1               40              30
2               8               2               70              62
```

11.5 Result

Executed memory allocation strategies successfully.

12 File Allocation Strategies

12.1 Problem

Simulate file allocation strategies. a) Sequential b) Indexed c) Linked.

12.2 Algorithm

12.2.1 Sequential

1.Start
2.Enter the user to enter the number of blocks.
3.Initialize all elements of the block array to 0.
4.As long as there are more files to allocate:
4.1.Set flag to 0.
4.2.Enter the starting block of the file b.
4.3.Enter the length of the file l.
4.4.If l>n then print "File cannot be allocated due to insufficient
space in the disk" and goto step 5
4.5.Else set k to 0.
4.6.Check if the current block is already allocated then set flag to 1
and break the loop.
4.7.If flag is 0 allocate the blocks.
4.8.print the allocated blocks.
5.Stop.

12.2.2 Indexed

1.Start
2.Enter the user to enter the number of blocks.
3.Initialize all elements of the block array to 0.
4.As long as there are more files to allocate:
4.1.Set count to 0.
4.2.Enter the index block number of the file index.
4.3.If index >= n || index < 0, then print "Invalid index block number
4.4.If the index block is already allocated print "Index block already allocated.
4.5.Enter the user to enter the length of the file.
4.6.Mark the index block as allocated.
4.7.Initialize start to 0.
4.8.while i<ptr->length:
a.if the current block is not allocated, set the block as allocated.
b.Increment count by 1.
4.9.If count <length,print "Insufficient storage! No allocation."
4.10.print the allocated blocks.
5.Stop.

12.2.3 Linked

```
1.Start
2.Enter the user to enter the number of blocks.
3.Initialize all elements of the block array to 0.
4.As long as there are more files to allocate:
4.1.Set flag to 0.
4.2.Enter the starting block of the file b.
4.3.Enter the length of the file l.
4.4.If starting block is already allocated then print "starting block is
allocated" and goto step 5
4.5.Else set k to 0.
4.6.Create a new block struct pointer b.
4.7.Set b->block_no to ptr->start_block.
4.8.Set b->next to NULL.
4.9.Set ptr->list to b.
4.10.Set blocks[ptr->start_block] to 1 to mark it as allocated.
4.11.Set count to 1.
4.12.Set i to ptr->start_block + 1.
4.13.If i<ptr->length:
a.If i>=n set i to 0
b.if the current block is not allocated create a new block
struct pointer and set new_block->block_no to i and
new_block->next to NULL.
c.assign the link and increment i and goto step 4.13
4.14.If count is less than ptr->length,print "Allocation not
possible due to insufficient storage!".
4.15.print the allocated blocks.
5.Stop.
```

12.3 Program Code

12.3.1 Sequential

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

void recurse(int files[]){
    int flag = 0, startBlock, len, j, k;
    char ch;
    printf("Enter the starting block and the length of the files: ");
    scanf("%d %d", &startBlock, &len);
    for (j=startBlock; j<(startBlock+len); j++){
        if (files[j] == 0)
            flag++;
    }
    if(len == flag){
        for (int k=startBlock; k<(startBlock+len); k++){
            if (files[k] == 0){
                files[k] = 1;
                printf("%d ", k);
```

```
            }
        }
        if (k != (startBlock+len-1))
            printf("\nThe file is allocated to the disk\n");
    }
    else
        printf("The file is not allocated to the disk\n");

    printf("Do you want to enter more files?\n");
    printf("Press y/n?: ");
    scanf(" %c", &ch);
    if (ch == 'y')
        recurse(files);
    else
        exit(0);
    return;
}

int main()
{
int files[500]={0};
for(int i=0;i<50;i++)
files[i]=0;
printf("Files Allocated are :\n");

recurse(files);
return 0;
}
```

12.3.2 Indexed

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
int n;
printf("\nNumber of blocks: ");
scanf("%d",&n);
int *blocks=(int*)malloc(sizeof(int));
for(int i=0;i<n;i++)
    blocks[i]=0;
char ch='y';
do
{
int index;
printf("\nIndex Block Number:");
scanf("%d",&index);
if(blocks[index]==1)
printf("\nIndex Full-No allocation possible");
else
```

```
{
int length;
printf("\nFile Length:");
scanf("%d",&length);
int count=0;
blocks[index]=1;
int indexblock[100];
for(int i=0;(i<n)&&(count<length);i++)
{
if(blocks[i]==0)
{
blocks[i]=1;
indexblock[count]=i;
count++;
}
}
if(count<length)
{
printf("\nInsufficient storage-No Allocation Possible");
for(int i=0;i<count;i++)
{
blocks[indexblock[i]]=0;
}
blocks[index]=0;
}
else
{
printf("\nAllocation:\n");
for(int i=0;i<count;i++)
printf("%d>%d\n",index,indexblock[i]);
}
}
printf("Continue y/n?: ");
    scanf(" %c", &ch);
}while(ch!='n');
}
```

12.3.3 Linked

```
#include<stdio.h>
#include<stdlib.h>
typedef struct block{
int blockno;
struct block *next;
}block;

typedef struct{
int startblock;
int length;
block *list;
}file;
```

```
void main()
{
int n;
printf("\nNumber of free blocks:");
scanf("%d",&n);
int blocks[100];
for(int i=0;i<n;i++)
blocks[i]=0;
char ch='y';
do
{
file *ptr=(file*)malloc(sizeof(file));
printf("\nStarting Block:");
scanf("%d",&(ptr->startblock));
printf("\nFile Length:");
scanf("%d",&(ptr->length));
if(blocks[ptr->startblock]==1)
printf("\nBlock already allocated");
else
{
block *b=(block*)malloc(sizeof(block));
b->blockno=ptr->startblock;
b->next=NULL;
ptr->list=b;
blocks[ptr->startblock]=1;
int count=1,i;
for(i=0;(i<n)&&(count<(ptr->length));i++)
{
if(blocks[i]==0)
{
block *new=(block*)malloc(sizeof(block));
new->blockno=i;
new->next=NULL;
block *p=ptr->list;
while(p->next!=NULL)
{
p=p->next;
}
p->next=new;
count++;
blocks[i]=1;
}
}
if(i==n&&count<(ptr->length))
{
printf("\nINSUFFICIENT STORAGE!");
}
else
{
printf("\nFile Allocation:\n");
block *p=ptr->list;
while(p->next!=NULL)
```

```
{
printf("%d->",p->blockno);
p=p->next;
}
printf("%d",p->blockno);
}
}
printf("\nContinue y/n?: ");
    scanf(" %c", &ch);
}while(ch!='n');
}
```

12.4 Sample Input and Output

Sequential

```
s21b46@administrator-rusa:~/OSL/exp11$ ./a.out
Input no. of blocks:3
Input no. of processes:3
Input block sizes:
block 1: 30

block 2: 70

block 3: 55

input process sizes:
process 1: 43

process 2: 50

process 3: 13

processs no.|process size|allocated block|allocated block size|internal fragmentation
1              43             2              70              27
2              50             3              55              5
3              13             1              30              17
s21b46@administrator-rusa:~/OSL/exp11$
```

Indexed

```
s21b46@administrator-rusa:~/OSL/exp11$ ./a.out
input the no. of blocks:3

input the no. of processes:3

input block sizes:
block 1: 50

block 2: 103

block 3: 201

input process sizes:
process 1: 101

process 2: 45

process 3: 150

processs no.|process size|allocated block|allocated block size|internal fragmentation
1              101            2              103             2
2              45             1              50              5
3              150            3              201             51
```

Linked

```
s21b46@administrator-rusa:~/OSL/11.Memory Allocation$ ./a.out
input the no. of blocks:3

input the no. of processes:3

input block sizes:
block 0:10

block 1:40

block 2:70

input process sizes:
process 0:5

process 1:10

process 2:8

processs no.|process size|allocated block|allocated block size|internal fragmentation
0               5               0               10              5
1               10              1               40              30
2               8               2               70              62
```

12.5 Result

Executed file allocation strategies successfully.

13 Disk Scheduling Algorithms

13.1 Problem

Simulate disk scheduling algorithms: a)FCFS b)SCAN c)C-SCAN

13.2 Algorithm

13.2.1 FCFS

1.Start
2.Input the number of requests to n.
3.Input requests to array req.
4.Input head position to variable head.
5.Iterate throught the req array and do the following:+
a. Print the current request value req[i].
b. Calculate the absolute difference between the current request and the head position add it to d.
c. Update the head position with the current request value.
6.display seek count d.
7.Stop

13.2.2 SCAN

1.Start
2.Input the number of requests to n.
3.Input requests to array req.
4.Input max track no and head position to variable max and head.
5.Sort the req[] array in ascending order.
6.Set l to n-1.
7.Iterate throught the req array and if the current request value is greater than or equal to the head position, set l to i and break the loop.
8.set d = max - head + max - req[0].
9.Use a loop to print the request values from l to n-1, representing the forward seek sequence.
a. Print the current request value req[i].
b. Update the head position with the current request value.
10.Use a loop to print the request values from l-1 to 1, representing the backward seek sequence.
a. Print the current request value req[i].
b. Update the head position with the current request value.
11.Print the first request value req[0].
12.display seek count d.
13.Stop

13.2.3 C-SCAN

1.Start

2.Input the number of requests to n.
3.Input requests to array req.
4.Input max track no and head position to variable max and head.
5.Sort the req[] array in ascending order.
6.Set l to n-1.
7.Iterate throught the req array and if the current request value is greater than or equal to the head position, set l to i and break the loop.
8.set d = max - head + max + req[l-1].
9.Use a loop to print the request values from l to n-1, representing the forward seek sequence.
a. Print the current request value req[i].
b. Update the head position with the current request value.
10.Print "0-->" to represent the seek to track 0.
11.Use a loop to print the request values from 0 to l-1, representing the backward seek sequence.
a. Print the current request value req[l-1].
b. Update the head position with the current request value.
12.Print the first request value req[0].
13.display seek count d.
14.Stop

13.3 Program Code

13.3.1 FCFS

```
#include<stdio.h>
#include<string.h>
void main()
{
int tr[20],cr,n,i,sum=0,new;
printf("ENTER NUMBER OF REQUESTS : ");
scanf("%d",&n);
printf("ENTER THE HEAD POSITION : ");
scanf("%d",&cr);
printf("ENTER THE INPUT ARRAY : ");
for(i=0;i<n;i++)
{
new = 0;
scanf("%d",&tr[i]);
new=cr-tr[i];
if(new<0)
{
new=tr[i]-cr;
}
cr=tr[i];
sum=sum + new;
}
printf("SEEK SEQUENCE : ");
for(i=0;i<n;i++)
printf("%d => ",tr[i]);
  printf("\b\b\b.   \nSEEK COUNT : %d\n",sum);
```

```
}
```

13.3.2 SCAN

```
#include <stdio.h>
#include <stdlib.h>
int pg[20];
int n;
void sort(){
int i,j,t;
for(i=0;i<n-1;i++)
for(j=0;j<n-i-1;j++){
if(pg[j]>pg[j+1]){
t=pg[j];
pg[j]=pg[j+1];
pg[j+1]=t;
}
}
}
int findloc(int head){
for(int i=0;i<n;i++){
if(head<pg[i])
return i;
}
}
void main(){
int head,i,j,k,max;
printf("Enter Number of Requests: ");
scanf("%d",&n);
printf("Enter Request Array: ");
for(i=0;i<n;i++)
scanf("%d",&pg[i]);
printf("Enter Head Position: ");
scanf("%d",&head);
printf("Enter max: ");
scanf("%d",&max);


printf("Seek Sequence: ");
if(pg[0]>head){
sort();
int pos=findloc(head);
// printf("\n%d\n",pos);
printf("%d ",head);
for(i=pos;i<n;i++){
printf("%d ",pg[i]);
}
printf("%d ",max);
for(i=pos-1;i>=0;i--){
printf("%d ",pg[i]);
```

```
}
printf("0\n");
}
else{
sort();
int pos=findloc(head);
// printf("\n%d\n",pos);
printf("%d ",head);
for(i=pos-1;i>=0;i--){
printf("%d ",pg[i]);
}
printf("0");
for(i=pos;i<n;i++){
printf("%d ",pg[i]);
}
printf("%d\n",max);

}

}
```

13.3.3 C-SCAN

```
#include <stdio.h>
#include <stdlib.h>
int pg[20];
int n;
void sort(){
int i,j,t;
for(i=0;i<n-1;i++)
for(j=0;j<n-i-1;j++){
if(pg[j]>pg[j+1]){
t=pg[j];
pg[j]=pg[j+1];
pg[j+1]=t;
}
}
}
int findloc(int head){
for(int i=0;i<n;i++){
if(head<pg[i])
return i;
}
}
void main(){
int head,i,j,k,max;
printf("Enter Number of Requests: ");
scanf("%d",&n);
printf("Enter Request Array: ");
for(i=0;i<n;i++)
scanf("%d",&pg[i]);
```

```
printf("Enter Head Position: ");
scanf("%d",&head);
printf("Enter max: ");
scanf("%d",&max);


printf("Seek Sequence: ");
if(pg[0]>head){
sort();
int pos=findloc(head);
// printf("\n%d\n",pos);
printf("%d ",head);
for(i=pos;i<n;i++){
printf("%d ",pg[i]);
}
printf("%d ",max);
printf("0 ");
for(i=0;i<pos;i++){
printf("%d ",pg[i]);
}
}
else{
sort();
int pos=findloc(head);
// printf("\n%d\n",pos);
printf("%d ",head);
for(i=pos-1;i>=0;i--){
printf("%d ",pg[i]);
}
printf("0 ");
printf("%d ",max);
for(i=n-1;i>pos;i--){
printf("%d ",pg[i]);
}
}
}
```

13.4 Sample Input and Output

FCFS

```
s21b46@administrator-rusa:~/OSL/exp13$ ./a.out
ENTER NUMBER OF REQUESTS : 8
ENTER THE HEAD POSITION : 50
ENTER THE INPUT ARRAY : 176 79 34 60 92 11 41 114
SEEK SEQUENCE : 176 => 79 => 34 => 60 => 92 => 11 => 41 => 114 .
SEEK COUNT : 510
```

SCAN

```
s21b46@administrator-rusa:~/OSL/exp13$ cc scan.c
s21b46@administrator-rusa:~/OSL/exp13$ ./a.out
Number of Requests:7

Request Array:82 170 43 140 24 16 190

Head Positon:50

Max Number:200

Seek Sequence:82 140 170 190 43 24 16
Seek Count:334
```

C-SCAN

```
s21b46@administrator-rusa:~/OSL/exp13$ ./a.out
inputtheno.ofrequests:7

inputrequestarray:82 170 43 140 24 16 190

inputcurrentheadposition:50

inputmaxtrackno.:200

seeksequenceis:82 140 170 190 16 24 43
seekcountis:393
```

13.5 Result

Executed Disk Scheduling Algorithms successfully.