RAMCO INSTITUTE OF TECHNOLOGY

Approved by AICTE, New Delhi & Affiliated to Anna University

Accredited by NAAC & An ISO 9001:2015 Certified Institution

NBA Accredited UG Programs: CSE, EEE, ECE and MECH

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

OPERATING SYSTEM LABORATORY

Subject Code: AL3452

II YEAR / IV SEMESTER

Prepared by	Approved by
Mrs.C.Karpagavalli, AP/AD	Dr.M.Kaliappan, HoD/AD

LIST OF EXPERIMENTS

1. Installation of Operating system : Windows/ Linux
2. Illustrate UNIX commands and Shell Programming
3. Process Management using System Calls : Fork, Exec, Getpid, Exit, Wait, Close
4. Write C programs to implement the various CPU Scheduling Algorithms
5. Illustrate the inter process communication strategy
6. Implement mutual exclusion by Semaphores
7. Write a C program to avoid Deadlock using Banker's Algorithm
8. Write a C program to Implement Deadlock Detection Algorithm
9. Write C program to implement Threading
10. Implement the paging Technique using C program
11. Write C programs to implement the following Memory Allocation Methods
    a. First Fit
    b. Worst Fit
    c. Best Fit
12. Write C programs to implement the various Page Replacement Algorithms
13. Write C programs to Implement the various File Organization Techniques
14. Implement the following File Allocation Strategies using C programs
    a. Sequential
    b. Indexed
    c. Linked
15. Write C programs for the implementation of various disk scheduling algorithms.

Exp.No : 1	**Installation of Operating System:Windows/Unix**
Date:	

Aim:

To Study the Installation of Operating System: Windows/Unix.

Procedure (To Install Windows):

The following step by step procedure will to install Windows XP.

Step1:

1. Connect the USB flash drive to your technician PC.
2. Open Disk Management: Right-click on Start and choose Disk Management.
3. Format the partition: Right-click the USB drives partition and chooses Format**.** Select the FAT32 file system to be able to boot either BIOS-based or UEFI-based PCs.
4. Set the partition as active: Right-click the USB drive partition and click Mark Partition as Active.

Step2:

5. Use File Explorer to copy and paste the entire contents of the Windows product DVD or ISO to the USB flash drive.
6. Optional: add an unattend file to automate the installation process. For more information, see Automate Windows Setup.

Step3:

7. Connect the USB flash drive to a new PC.
8. Turn on the PC and press the key that opens the boot-device selection menu for the computer, such as the Esc/F10/F12 keys. Select the option that boots the PC from the USB flash drive. Windows Setup starts. Follow the instructions to install Windows.
9. Remove the USB flash drive.

Windows USB install drives are formatted as FAT32, which has a 4GB filesize limit. If your image is larger than the filesize limit:

1. Copy everything except the Windows image file (sources\install.wim) to the USB drive (either drag and drop, or use this command, where D: is the mounted ISO and E: is the USB flash drive.)

   Command:

   ```
   Robocopy D: E: /s/max: 380000000
   ```

2. Split the Windows image file into smaller files, and put the smaller files onto the USB drive:

   Command:

```
Dism /Split-Image /ImageFile:D:\sources\install.wim /SWMFile:E:\sources\install.swm
/FileSize: 3800
```

Procedure (To Install Linux):

The following step by step procedure will to install Linux.

Step1:

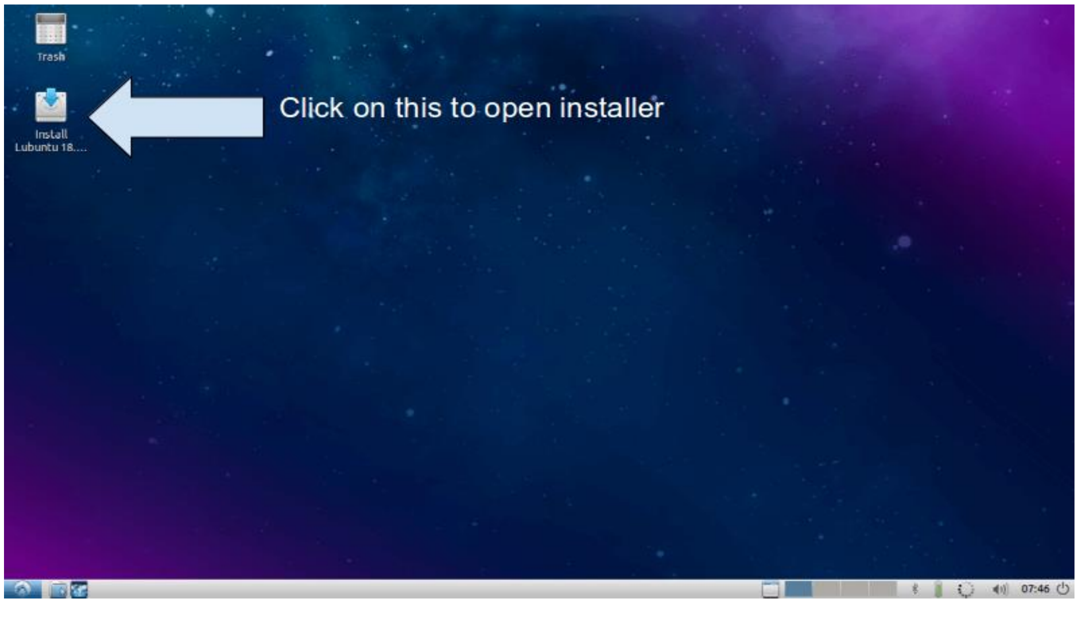
1. Use your Linux ISO image file to create a bootable USB installation media. You can use any software like Unetbootin, Gnome Disk Utility, Yumi Multi Boot, xboot, Live USB Creator, etc. to create bootable USB with the help of ISO image file.

Step2: The two main partitions are there.

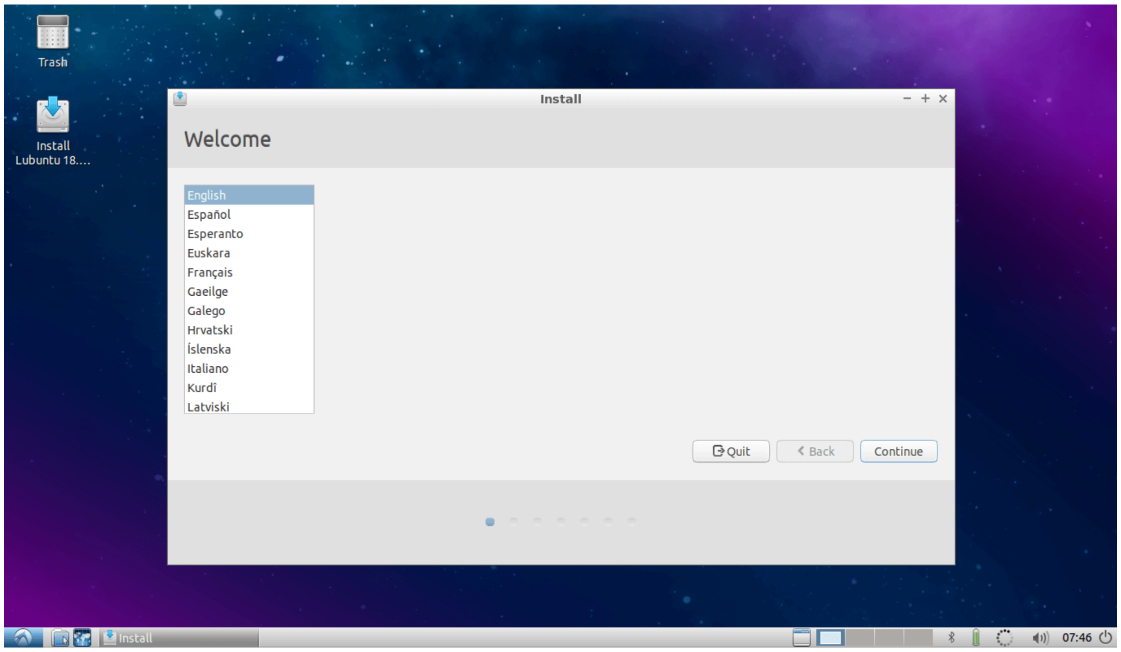
1. The root partition of format ext4 of size according to your use.
2. Optionally you can use the rest of the space as a FAT partition for using it as a normal USB drive.

Step3:

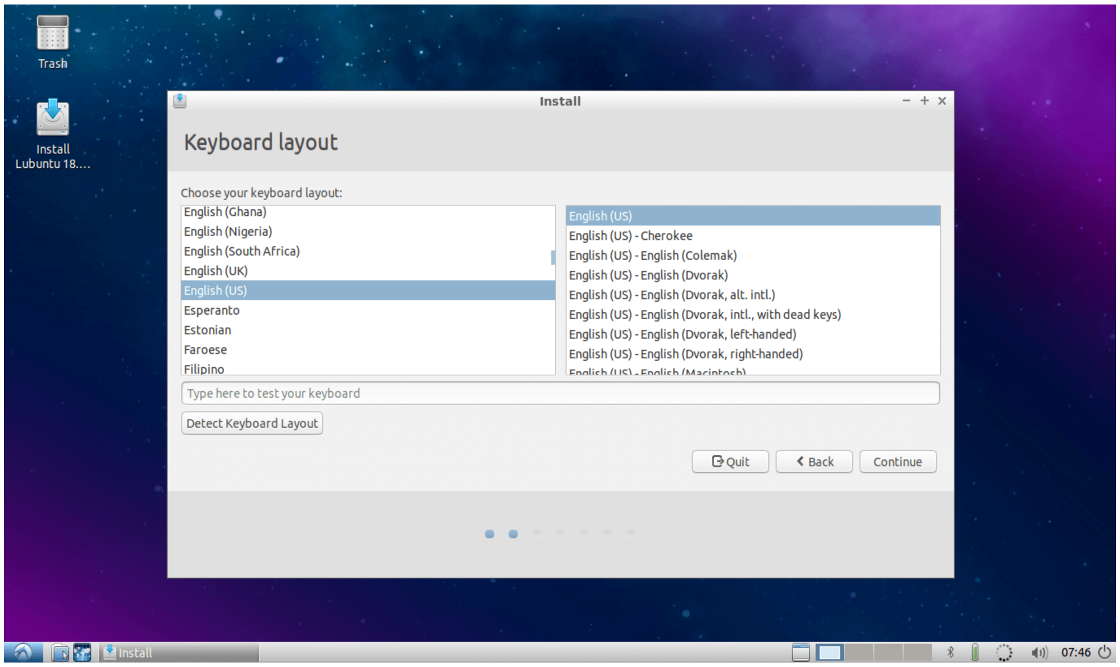
1. First, boot Linux OS (Lubuntu 18.04) from your bootable installation media and launch installation application from a live session.



2. Installer welcome screen will appear, select Language there and hit Continue.

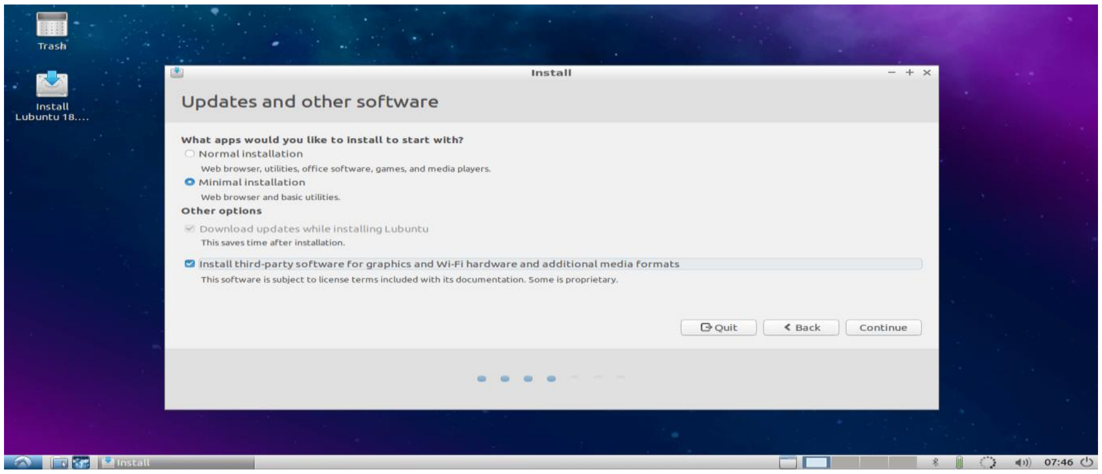


3. Select Keyboard Layout and continue


Trash
Install Lubuntu 18....
Install
Keyboard layout
Choose your keyboard layout:
English (Ghana)
English (Nigeria)
English (South Africa)
English (UK)
English (US)
Esperanto
Estonian
Faroese
Filipino
English (US)
English (US) - Cherokee
English (US) - English (Colemak)
English (US) - English (Dvorak)
English (US) - English (Dvorak, alt. intl.)
English (US) - English (Dvorak, intl., with dead keys)
English (US) - English (Dvorak, left-handed)
English (US) - English (Dvorak, right-handed)
Type here to test your keyboard
Detect Keyboard Layout
Quit
Back
Continue
07:46

4. Select Wifi internet if you want to update Lubuntu while installation.

5. Select Installation Type and Third-party installation as per your choice and go to next.



Updates and other software

What apps would you like to install to start with?

( ) Normal installation
Web browser, utilities, office software, games, and media players.

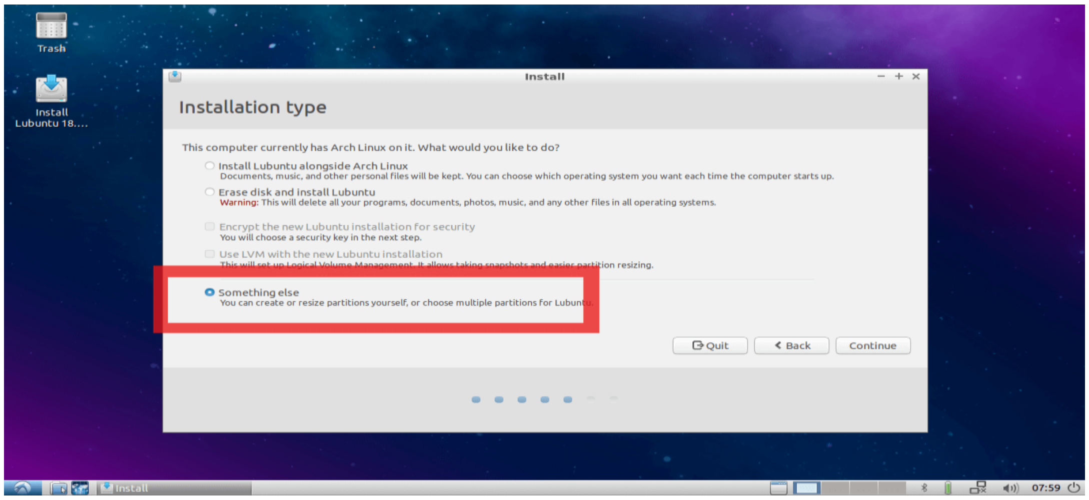
(●) Minimal installation
Web browser and basic utilities.

Other options

[✓] Download updates while installing Lubuntu
This saves time after installation.

[✓] Install third-party software for graphics and Wi-Fi hardware and additional media formats
This software is subject to license terms included with its documentation. Some is proprietary.

6. Here select Something Else Option (It is Mandatory) and go to next…



Installation type

This computer currently has Arch Linux on it. What would you like to do?

( ) Install Lubuntu alongside Arch Linux
Documents, music, and other personal files will be kept. You can choose which operating system you want each time the computer starts up.

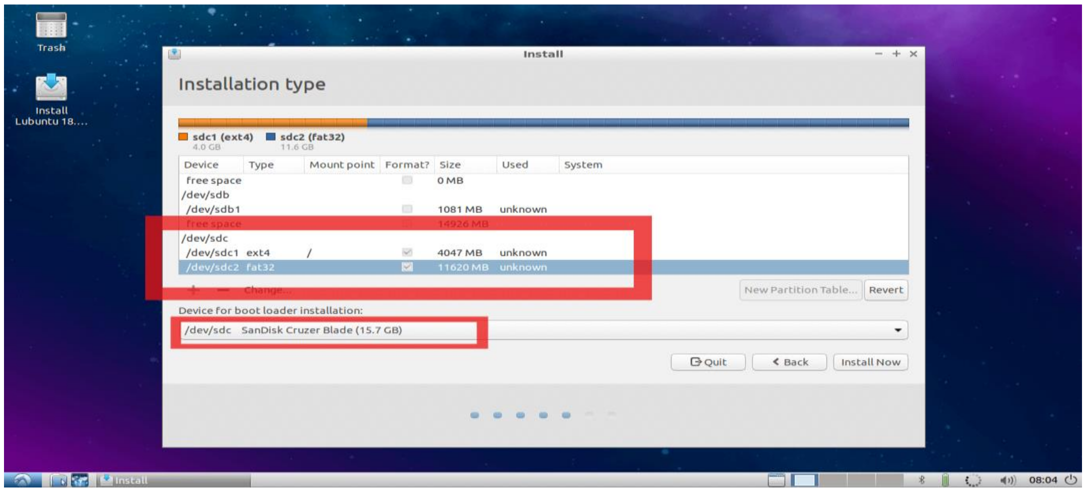
( ) Erase disk and install Lubuntu
Warning: This will delete all your programs, documents, photos, music, and any other files in all operating systems.

[ ] Encrypt the new Lubuntu installation for security
You will choose a security key in the next step.

[ ] Use LVM with the new Lubuntu installation
This will set up Logical Volume Management. It allows taking snapshots and easier partition resizing.

(●) Something else
You can create or resize partitions yourself, or choose multiple partitions for Lubuntu.

7. This is an important step; here you need to find out where your Main USB drive is mounted.



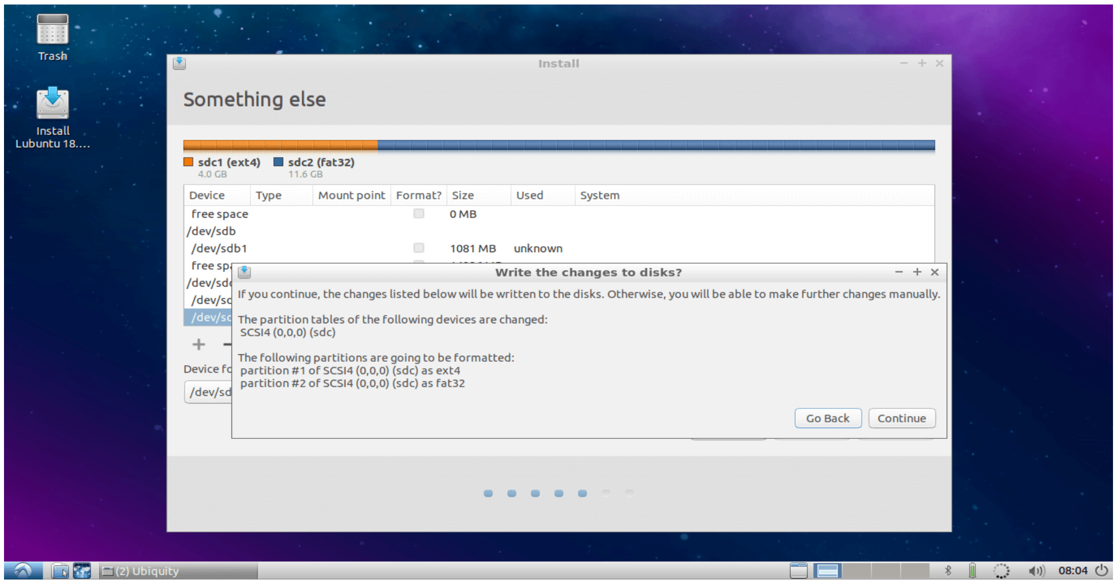
Installation type

🟧 sdc1 (ext4) 4.0 GB 🟦 sdc2 (fat32) 11.6 GB

Device	Type	Mount point	Format?	Size	Used	System
free space			☐	0 MB		
/dev/sdb						
/dev/sdb1			☐	1081 MB	unknown	
free space			☐	14926 MB		
/dev/sdc						
/dev/sdc1	ext4	/	✓	4047 MB	unknown	
/dev/sdc2	fat32		✓	11620 MB	unknown	

Device for boot loader installation:
/dev/sdc SanDisk Cruzer Blade (15.7 GB)

8. Make sure that the device and drives shown on this window are of your Main USB drive, which is in my case /dev/sdc. Hit continue…



Something else

🟧 sdc1 (ext4) 4.0 GB 🟦 sdc2 (fat32) 11.6 GB

Device	Type	Mount point	Format?	Size	Used	System
free space			☐	0 MB		
/dev/sdb						
/dev/sdb1			☐	1081 MB	unknown	

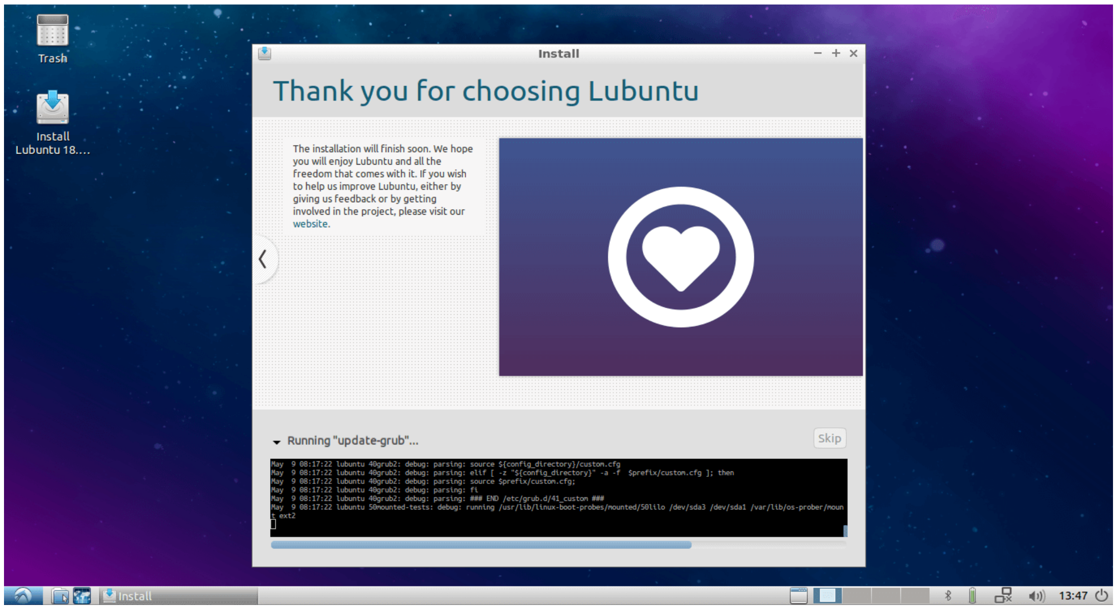
Write the changes to disks?

If you continue, the changes listed below will be written to the disks. Otherwise, you will be able to make further changes manually.

The partition tables of the following devices are changed:
SCSI4 (0,0,0) (sdc)

The following partitions are going to be formatted:
partition #1 of SCSI4 (0,0,0) (sdc) as ext4
partition #2 of SCSI4 (0,0,0) (sdc) as fat32

9. Now select your Region and hit Continue…
10. Add username, password, and hostname, etc…
11. Let the installation finish.



Thank you for choosing Lubuntu

The installation will finish soon. We hope you will enjoy Lubuntu and all the freedom that comes with it. If you wish to help us improve Lubuntu, either by giving us feedback or by getting involved in the project, please visit our website.

Running "update-grub"...

```
May  9 08:17:22 lubuntu 40grub2: debug: parsing: source ${config_directory}/custom.cfg
May  9 08:17:22 lubuntu 40grub2: debug: parsing: elif [ -z "${config_directory}" -a -f  $prefix/custom.cfg ]; then
May  9 08:17:22 lubuntu 40grub2: debug: parsing: source $prefix/custom.cfg;
May  9 08:17:22 lubuntu 40grub2: debug: parsing: fi
May  9 08:17:22 lubuntu 40grub2: debug: parsing: ### END /etc/grub.d/41_custom ###
May  9 08:17:22 lubuntu 50mounted-tests: debug: running /usr/lib/linux-boot-probes/mounted/50lilo /dev/sda3 /dev/sda1 /var/lib/os-prober/moun
t ext2
```

12. After completing installation hit restart and remove your installation media and press Enter.

13. Congratulations, you have successfully installed your own Linux OS on your pen drive to use it on any PC. Now you can connect a USB drive to any PC and start your system on that PC by simply selecting boot from USB option while booting.

Result:

Thus the study experiment installation procedure of opearitng Systems: Window/Linux has been Successfully Completed.

Exp.No : 2	**Illustrate UNIX commands and Shell Programming**
Date:	

Aim:

To Illustrate the UNIX Commands and Shell Programming.

i)Procedure (UNIX Commands):

Steps to Login:

1. Type **telnet** *server_ipaddress* in **run** window.
2. User has to authenticate himself by providing *username* and *password*. Once verified, a greeting and **$** prompt appears. The shell is now ready to receive commands from the user. Options suffixed with a hyphen (–) and arguments are separated by space.

General commands

Command	**Function**
Date	Used to display the current system date and time.
date +%D	Displays date only
date +%T	Displays time only
date +%Y	Displays the year part of date
date +%H	Displays the hour part of time
Cal	Calendar of the current month
cal*year*	Displays calendar for all months of the specified year
cal*month year*	Displays calendar for the specified month of the year
Who	Login details of all users such as their IP, Terminal No, User name,
who am i	Used to display the login details of the user
Uname	Displays the Operating System
uname –r	Shows version number of the OS (kernel).
uname –n	Displays domain name of the server
echo$HOME	Displays the user's home directory
Bc	Basic calculator. Press Ctrl+dto quit
lp *file*	Allows the user to spool a job along with others in a print queue.
man*cmdname*	Manual for the given command. Press qto exit
history	To display the commands used by the user since log on.
exit	Exit from a process. If shell is the only process then logs out

Directory commands

Command	Function
Pwd	Path of the present working directory
mkdir*dir*	A directory is created in the given name under the current directory
mkdir*dir1 dir2*	A number of sub-directories can be created under one stroke
cd*subdir*	Change Directory. If the *subdir* starts with **/** then path starts from **root** (absolute) otherwise from current working directory.
Cd	To switch to the home directory.
cd /	To switch to the root directory.
cd ..	To move back to the parent directory
rmdir*subdir*	Removes an empty sub-directory.

File commands

Command	Function
cat >*filename*	To create a file with some contents. To end typing press Ctrl+d. The **>** symbol means redirecting output to a file. (< for input)
cat*filename*	Displays the file contents.
cat>>*filename*	Used to append contents to a file
cp*src des*	Copy files to given location. If already exists, it will be overwritten
cp –i*src des*	Warns the user prior to overwriting the destination file
cp –r *src des*	Copies the entire directory, all its sub-directories and files.
mv *old new*	To rename an existing file or directory. –i option can also be used
mv *f1 f2 f3 dir*	To move a group of files to a directory.
mv –v *old new*	Display name of each file as it is moved.
rm *file*	Used to delete a file or group of files. –i option can also be used
rm *	To delete all the files in the directory.
rm –r *	Deletes all files and sub-directories
rm –f *	To forcibly remove even write-protected files
Ls	Lists all files and subdirectories (blue colored) in sorted manner.
ls*name*	To check whether a file or directory exists.
ls*name******	Short-hand notation to list out filenames of a specific pattern.
ls –a	Lists all files including hidden files (files beginning with **.**)
ls –x*dirname*	To have specific listing of a directory.
ls –R	Recursive listing of all files in the subdirectories
ls –l	Long listing showing file access rights (read/write/execute-**rwx** for user/group/others-**ugo**).
cmp *file1 file2*	Used to compare two files. Displays nothing if files are identical.
wc *file*	It produces a statistics of lines (**l**), words(**w**), and characters(**c**).
chmod *perm file*	Changes permission for the specified file. (r=4, w=2, x=1) chmod 740 *file* sets all rights for user, read only for groups and no rights for others

3. The commands can be combined using the pipeline (|) operator. For example, number of users logged in can be obtained as.

```
who | wc –l
```

4. Finally to terminate the UNIX session execute the command **exit** or **logout**.

Output

```
$ date
       Sat Apr     9 13:03:47 IST 2011

$ date +%D

        04/09/11

$ date +%T
        13:05:33

$ date +%Y
        2011

$ date +%H
        13

$ cal 08 1998
        August 1998

Su Mo Tu We Th Fr Sa

1

2 3   4   5   6   7   8

9 10 11 12 13 14 15

16 17 18 19 20 21 22

30 31

$ who

       Root:0      Apr  9 08:41

       vijai       pts/0 Apr   9 13:00 (scl-64)
```

```
            ad3025       pts/3 Apr    9 13:18 (scl-41.smkfomra.com)
$ uname

            Linux

$ uname –r
            2.4.20-8smp

$ uname –n
            localhost.localdomain

$ echo $HOME
            /home/vijai

$ echo $USER
            Vijai

$ bc

            3+5

            8

$ pwd

            /home/vijai/shellscripts/loops

$ mkdir filter
$ ls

            filter    list.sh   regexpr shellscripts

$ cd shellscripts/loop

$

$ cd

$

$ cd /

            [vijai@localhost /]$

[vijai@localhost /]$ cd /home/vijai/shellscripts/loops/
```

```
$ cd ..

              [vijai@localhost shellscripts]$

$ rmdir filter
$ ls

            list.sh    regexpr   shellscripts

$ cat > greet
            hi AI
           wishing u the best

$ cat >> greet
           Bye

$ cat greet
              hi AI
             wishing u the best bye.
$ ls

             greet   list.sh   regexpr  shellscripts

$ ls –a

         .          .bash_logout   .canna     .gtkrc     regexpr     .viminfo.tmp
        ..          .bash_profile  .emacs      .kde        shellscripts  .xemacs
  .bash_history      .bashrc       greet      list.sh               .viminfo

$ ls –l

  -rw-rw-r--   1   Vijai      vijai      32     Apr    11   14:52    greet
  -rw-rw-r--   1   Vijai      vijai      30     Apr     4   13:58    list.sh
 drwxrwxr-x   2   Vijai      vijai     4096    Apr     9   14:30   regexpr

$ cp greet ./regexpr/
$ ls

                   greet     list.sh    regexpr     shellscripts

$ ls ./regexpr
                   demo greet
```

```
$ cp -i greet ./regexpr/
cp: overwrite 'greet'? n


$ mv greet greet.txt
$ ls

                 greet.txt  list.sh     regexpr     shellscripts

$ mv greet.txt ./regexpr/
$ ls

               list.sh       regexpr  shellscripts

$ rm -i *.sh
       rm: remove regular file 'fact.sh'? y rm:
       remove regular file 'prime.sh'?y

$ ls

               list.sh       regexpr  shellscripts

$ wc list.sh
              4        9       30 list.sh

$ wc -l list.sh
              4 list.sh

$ cmp list.sh fact.sh
               list.sh fact.sh differ: byte 1, line 1

$ ls -l list.sh
              -rw-rw-r--  1   vijai  vijai  30 Apr4 13:58 list.sh

$ chmod ug+x list.sh

$ ls -l list.sh
               -rwxrwxr--        1 vijai   vijai     30 Apr   4 13:58 list.sh

$ chmod 740 list.sh
$ ls -l list.sh
                -rwxr-----         1 vijai   vijai     30 Apr   4 13:58 list.sh
```

ii) Procedure (Shell Programming):

The activities of a shell are not restricted to command interpretation alone. The shell also has rudimentary programming features. Shell programs are stored in a file (with extension .**sh**). Shell programs run in interpretive mode. The original UNIX came with the Bourne shell (**sh**) and it is universal even today. C shell (**csh**) and Korn shell (**ksh**) are also widely used. Linux offers Bash shell (**bash**) as a superior alternative to Bourne shell.

Preliminaries

1. Comments in shell script start with **#**.
2. Shell variables are loosely typed i.e. not declared. Variables in an expression or output must be prefixed by **$**.
3. The **read** statement is shell's internal tool for making scripts interactive.
4. Output is displayed using **echo**statement
5. Expressions are computed using the **expr** command. Arithmetic operators are + - * / %. Meta characters *** ( )** should be escaped with a \\.
6. The shell scripts are executed
   $ sh *filenams*

Multi-way branching

The case statement is used to compare a variables value against a set of constants. If it matches a constant, then the set of statements followed after) is executed till a; is encountered. The optional *default* block is indicated by *****. Multiple constants can be specified in a single pattern separated by **|**.

```
casevariable in
constant1)

statements ;;
constant2)
statements ;;
. . .
*)
statements
esac
```

Decision-making

Shell supports decision-making using **if** statement. The **if** statement like its counterpart in programming languages has the following formats.

if [*condition*] then	if [*condition*] then	if [*condition*] then
statements	*statements*	*statements*
fi	else	elif [*condition*] then
	statements	*statements*
	fi	……
		else
		statements
		fi

The set of relational operators are –eq –ne –gt –ge –lt –le and logical operators used in conditional expression are –a –o!

Loops

Shell supports a set of loops such as **for**, **while** and **until** to execute a set of statements repeatedly. The body of the loop is contained between **do** and **done** statement. The **for** loop doesn't test a condition, but uses a list instead.

for*variable* in *list*

do

statements

done

a) The **While** loop executes the *statements* as long as the condition remains true.

while [*condition*]

do

statements

done

b) The **Until** loop complements the while construct in the sense that the *statements* are executed as long as the condition remains false.

Until [*condition*]

do

Statements

done

A) Swapping values of two variables

```
# Swapping values – swap.sh echo -n
            "Enter value for A : " read a
echo -n   "Enter value for B : " read b
t=$a
a=$b
b=$t
echo "Values after Swapping"
echo "A Value is $a and B Value is $b"
```

Output

```
$ sh swap.sh
Enter value for A: 12 Enter
value for B: 23 Values after
Swapping

A Value is 23 and B Value is 12
```

B) Farenheit to Centigrade Conversion

```
# Degree conversion – degconv.sh echo -n
"Enter Fahrenheit : " read f

c=`expr \( $f - 32 \) \* 5 / 9` echo "Centigrade
is : $c"
```

Output

```
$ sh degconv.sh
Enter Fahrenheit: 213
Centigrade is: 100
```

C) Biggest of 3 numbers

```
# Biggest – big3.sh

echo -n "Give value for A B and C: " read a b c

if [ $a -gt $b -a $a -gt $c ] then

    echo "A is the Biggest number" elif [ $b –gt $c ]

then

    echo "B is the Biggest number" else

    echo "C is the Biggest number"

fi
```

Output

```
$ sh big3.sh
Give value for A B and C: 4 3 4 C is the
Biggest number
```

D) Grade Determination

```
# Grade – grade.sh

echo -n "Enter the mark : " read mark

if [ $mark -gt 90 ] then

    echo "S Grade" elif [
$mark -gt 80 ] then

    echo "A Grade" elif [
$mark -gt 70 ] then

    echo "B Grade" elif [
$mark -gt 60 ] then

    echo "C Grade" elif [
$mark -gt 55 ] then

    echo "D Grade" elif [
$mark -ge 50 ] then

    echo "E Grade" else

    echo "U Grade"

fi
```

Output

```
$ sh grade.sh Enter the
mark : 65 C Grade
```

E) Vowel or Consonant

```
# Vowel     - vowel.sh

echo -n "Key in a lower case character : " read choice
```

```
  case $choice in

          a|e|i|o|u) echo "It's a Vowel";; *)

          echo "It's a Consonant"

esac
```

F) **Simple Calculator**

```
# Arithmetic operations — calc.sh echo -n
"Enter the two numbers : " read a b

  echo " 1. Addition" echo " 2.
  Subtraction"

  echo " 3. Multiplication" echo " 4.
  Division"

  echo -n "Enter the option : " read option

  case $option in

      1) c=`expr $a +  $b` echo "$a +
         $b =  $c";;
      2) c=`expr $a -  $b` echo "$a -
         $b =  $c";;
      3) c=`expr $a \* $b` echo "$a *
         $b =  $c";;
      4) c=`expr $a /  $b` echo "$a /
         $b =  $c";;
      *) echo "Invalid Option" esac
```

Output

```
$ sh calc.sh
Enter the two numbers : 2 4

 1. Addition
 2. Subtraction
 3. Multiplication
 4. Division
Enter the option : 1 2 + 4 = 6
```

G) Multiplication Table

```
# Multiplication table – multable.sh clear
```

```
 echo -n "Which multiplication table?: " read n


for x in 1 2 3 4 5 6 7 8 9 10 do
      p=`expr $x \* $n`
      echo -n "$n X $x = $p" sleep 1
done
```

Output

```
$ sh multable.sh
Which multiplication table? : 6 6 X 1 = 6
     6   X  2 = 12
```

H) Number Reverse

```
# To reverse a number – reverse.sh echo -n
"Enter a number : "
read n rd=0
while [ $n -gt 0 ] do
      rem=`expr $n % 10` rd=`expr $rd
      \* 10 + $rem` n=`expr $n / 10`
done
echo "Reversed number is $rd"
```

Output

```
$ sh reverse.sh
Enter a number : 234 Reversed
number is  432
```

I) Prime Number

```
# Prime number – prime.sh echo -n
"Enter the number : " read n
i=2
```

```
m=`expr $n / 2` until [ $i -
gt $m ] do

     q=`expr $n % $i` if [ $q
     -eq 0 ] then

          echo "Not a Prime number" exit

     fi

     i=`expr $i + 1` done

echo "Prime number"
```

Output

```
$ sh prime.sh
Enter the number : 17 Prime
number
```

Result

Thus the various UNIX commands and shell scripts were executed using different programming constructs.

Exp.No : 3	**Process Management using System Calls : Fork, Exec, Getpid, Exit, Wait, Close**
Date:	

Aim:

To create a new child process using fork, Exec, Getpid, Exit, Wait, Close.

Procedure:

i) fork ()

- The fork system call is used to create a new process called *child* process.
    - The return value is 0 for a child process.
    - The return value is negative if process creation is unsuccessful.
    - For the parent process, return value is positive
- The child process is an exact copy of the parent process.
- Both the child and parent continue to execute the instructions following fork call.

The child can start execution before the parent or vice-versa.

ii) getpid()and getppid()

- The getpid system call returns process ID of the calling process
- The getppid system call returns parent process ID of the calling process

Algorithm

```
1. Declare a variable x to be shared by both child and parent.
2. Create a child process using fork system call.
3. If return value is -1 then
       Print "Process creation unsuccessfull"
       Terminate using exit system call.
4. If return value is 0 then
       Print "Child process"
       Print process id of the child using getpid system
       call Print value of x
       Print process id of the parent using getppid system call
5. Otherwise
```

```
        Print "Parent process"
        Print process id of the parent using getpid system call Print value of x
        Print process id of the shell using getppid system call.
6. Stop
```

Program

```
#include <stdio.h>
#include <stdlib.h>
#include<unistd.h>
#include<sys/types.h>
main()
 {
     pid_t    pid;
      int x = 5;
     pid = fork();
     x++;

     if (pid < 0)
     {
          printf("Process creation error"); exit(-1);
     }
     else if (pid == 0)
     {
          printf("Child process:"); printf("\nProcess id is %d",
          getpid()); printf("\nValue of x is %d",  x);
          printf("\nProcess id of parent is %d\n",  getppid());
     }
     else
     {
         printf("\nParent   process:");   printf("\nProcess   id   is
         %d", getpid()); printf("\nValue of x is %d",  x);
          printf("\nProcess id of shell is %d\n",  getppid());
     }

 }
```

Output

```
$ gcc fork.c
$. /a.out Child process:
Process id is 19499 Value
of x is 6
Process id of parent is 19498
Parent process: Process id
is 19498 Value of x is 6
Process id of shell is 3266
```

iii) exec()

- The exec family of function (execl, execv, execle, execve, execlp, execvp) is used by the child process to load a program and execute.
- execl system call requires path, program name and null pointer

Algorithm

```
1. Create a child process using fork system call.
2. If return value is -1 then
   a. Print "Process creation unsuccessfull"
3. Terminate using exit system call.
4. If return value is > 0 then
   a. Suspend parent process until child completes using wait system call
   b. Print "Child Terminated".
   c. Terminate the parent process.
5. If return value is 0 then
   a. Print "Child starts"
   b. Load date program into child process using exec system call.
   c. Terminate the child process.
6. Stop
```

Program

```
/* Load a program in child process - exec.c */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
main()
{
    pid_t pid;
    switch(pid = fork())
    {
        case -1:
            perror("Fork failed"); exit(-1);

        case 0:
            printf("Child     process\n");
            execl("/bin/date","date",0);
            exit(0);

        default:
            wait(NULL);
            printf("Child Terminated\n");
            exit(0);
    }
}
```

Output

```
$ gcc exec.c
$ ./a.out
Child process
Sat Feb 23 17:46:59 IST 2013
Child Terminated
```

iv) wait()

- The wait system call causes the parent process to be blocked until a child terminates.
- When a process terminates, the kernel notifies the parent by sending the SIGCHLD signal to the parent.
- Without wait, the parent may finish first leaving a *zombie* child, to be adopted by init process

Algorithm

```
1. Create a child process using fork system call.
2. If return value is -1 then
    a. Print "Process creation unsuccessfull"
3. Terminate using exit system call.
4. If return value is > 0 then
    a. Suspend parent process until child completes using wait system call
    b. Print "Parent starts"
    c. Print even numbers from 0–10
    d. Print "Parent ends"
5. If return value is 0 then
    a. Print "Child starts"
    b. Print odd numbers from 0–10
    c. Print "Child ends"
6. Stop
```

7. Program

/* Wait for child termination - wait.c */

```
#include <stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include <sys/wait.h>
main()
{
int i, status; pid_t pid;
pid = fork();
if (pid < 0)
{
printf("\nProcess creation failure\n"); exit(-1);
}
else if(pid > 0)
{
wait(NULL);
printf ("\nParent starts\nEven Nos: ");
for (i=2;i<=10;i+=2)
printf ("%3d",i);
printf ("\nParent ends\n");
}
else if (pid == 0)
{
printf ("Child starts\nOdd Nos: ");
for (i=1;i<10;i+=2)
printf ("%3d",i); printf ("\nChild  ends\n");
}
}
```

Output

```
$ gcc  wait.c

$. /a.out Child  starts
Odd Nos:     1    3    5    7    9
Child ends

Parent starts
Even Nos:      2    4    6    8    10
Parent ends
```

V) Exit ()

- The exit system call is used to terminate a process either normally or abnormally
- Closes all standard I/O streams.

Stat ()

- ➢ The stat system call is used to return information about a file as a structure.

Algorithm

1. Get *filename* as command line argument.
2. If *filename* does not exist then stop.
3. Call stat system call on the *filename* that returns a structure
4. Display members st_uid, st_gid, st_blksize, st_block, st_size, st_nlink, etc.,
5. Convert time members such as st_atime, st_mtime into time using ctime function
6. Compare st_mode with mode constants such as S_IRUSR, S_IWGRP, S_IXOTH and display file permissions.
7. Stop

Program

```
/* File status - stat.c */

#include <stdio.h>

#include<sys/stat.h>

#include<stdlib.h>

#include <time.h>
int main(int argc, char*argv[])
```

```
{
    struct stat file; int n;
    if (argc != 2)
    {
        printf("Usage: ./a.out <filename>\n"); exit(-1);
    }
    if ((n = stat(argv[1], &file)) == -1)
    {
        perror(argv[1]);
        exit(-1);
    }
    printf("User id : %d\n", file.st_uid);
    printf("Group id : %d\n", file.st_gid);
    printf("Block size : %d\n", file.st_blksize);
    printf("Blocks allocated : %d\n", file.st_blocks);
    printf("Inode no. : %d\n",  file.st_ino);
    printf("Last accessed : %s", ctime(&(file.st_atime)));
    printf("Last modified : %s", ctime(&(file.st_mtime)));
    printf("File size : %d bytes\n", file.st_size);
    printf("No. of links : %d\n",  file.st_nlink);
    printf("Permissions : ");
    printf( (S_ISDIR(file.st_mode)) ? "d" : "-");
    printf( (file.st_mode & S_IRUSR) ? "r" :  "-");
    printf( (file.st_mode & S_IWUSR) ? "w" :  "-");
    printf( (file.st_mode & S_IXUSR) ? "x" :  "-");
    printf( (file.st_mode & S_IRGRP) ? "r" :  "-");
    printf( (file.st_mode & S_IWGRP) ? "w" :  "-");
    printf( (file.st_mode & S_IXGRP) ? "x" :  "-");
    printf( (file.st_mode & S_IROTH) ? "r" :  "-");
    printf( (file.st_mode & S_IWOTH) ? "w" :  "-");
    printf( (file.st_mode & S_IXOTH) ? "x" : "-"); printf("\n");
    if(file.st_mode & S_IFREG) printf("File type :
        Regular\n");
    if(file.st_mode  &  S_IFDIR)  printf("File  type  :
        Directory\n");
}
```

Output

```
$ gcc stat.c
$ ./a.out fork.c
User id :  0 Group id : 0
Block size :  4096
Blocks allocated : 8 Inode no. :  16627
Last accessed : Fri Feb 22 21:57:09 2013
Last modified : Fri Feb 22 21:56:13 2013 File size : 591 bytes
No. of links : 1 Permissions : -
rw-r--r-- File type : Regular
```

Result:

Thus the above new child process using fork, Exec, Getpid, Exit, Wait, Close has been successfully completed.

Exp.No : 4	**Write C programs to implement the various CPU Scheduling Algorithms**
Date:	

Aim:

To schedule snapshot of processes queued according to FCFS, SJF, Priority, Round Robin scheduling algorithms.

Procedure:

i) FCFS Scheduling:

Process Scheduling

- CPU scheduling is used in multiprogrammed operating systems.
- By switching CPU among processes, efficiency of the system can be improved.
- Some scheduling algorithms are FCFS, SJF, Priority, Round-Robin, etc.
- Gantt chart provides a way of visualizing CPU scheduling and enables to understand better.

First Come First Serve (FCFS)

- Process that comes first is processed first
- FCFS scheduling is non-preemptive
- Not efficient as it results in long average waiting time.
- Can result in starvation, if processes at beginning of the queue have long bursts.

Algorithm

1. Define an array of structure *process* with members *pid*, *btime*, *wtime* & *ttime*.
2. Get length of the ready queue, i.e., number of process (say *n*)
3. Obtain *btime* for each process.
4. The *wtime* for first process is 0.
5. Compute *wtime* and *ttime* for each process as:
   a. $wtime_{i+1} = wtime_i + btime_i$
   b. $ttime_i = wtime_i + btime_i$
6. Compute average waiting time *awat* and average turnaround time *atur*

7. Display the *btime*, *ttime* and *wtime* for each process.
8. Display GANTT chart for the above scheduling
9. Display *awat* time and *atur*
10. Stop

Program

```
/* FCFS Scheduling      - fcfs.c */

 #include <stdio.h>
struct process
{
     int pid; int
     btime; int
     wtime; int
     ttime;
} p[10];
main()
{
     int i,j,k,n,ttur,twat;
     float awat,atur;

     printf("Enter no. of process : ");
     scanf("%d", &n);
     for(i=0; i<n; i++)
     {
          printf("Burst time for process P%d (in ms) : ",(i+1));
          scanf("%d", &p[i].btime);
          p[i].pid =  i+1;
     }
     p[0].wtime = 0; for(i=0; i<n;  i++)
     {
          p[i+1].wtime = p[i].wtime + p[i].btime;
          p[i].ttime = p[i].wtime +  p[i].btime;
     }
     ttur = twat = 0;
     for(i=0; i<n;  i++)
     {
          ttur += p[i].ttime;
          twat +=  p[i].wtime;
     }
     awat = (float)twat / n;
```

```
atur = (float)ttur /  n;
printf("\n FCFS Scheduling\n\n");
 for(i=0; i<28; i++)
     printf("-");
printf("\nProcess B-Time T-Time W-Time\n");
for(i=0; i<28; i++)
printf("-");
for(i=0; i<n; i++)
     printf("\n
                    P%d\t%4d\t%3d\t%2d",p[i].pid,p[i].btime,p[i].ttime,p[i
               ].wtime);
     printf("\n");
     for(i=0; i<28;  i++)
     printf("-");
     printf("\n\nAverage  waiting  time    : %5.2fms", awat);
     printf("\nAverage turn around time : %5.2fms\n", atur);
     printf("\n\nGANTT
     Chart\n"); printf("-");
     for(i=0; i<(p[n-1].ttime + 2*n); i++)
           printf(" ");
           printf("\n");
           printf("|");
   for(i=0; i<n;  i++)
  {
     k = p[i].btime/2;
     for(j=0; j<k;  j++)
          printf(" "); printf("P%d",p[i].pid);
     for(j=k+1; j<p[i].btime; j++)
          printf(" ");
          printf("|");
}
printf("\n");
printf("-");
for(i=0; i<(p[n-1].ttime + 2*n); i++)
  printf("-");
  printf("\n");
  printf("0");
  for(i=0; i<n;  i++)
  {
     for(j=0; j<p[i].btime; j++)
          printf(" ");
     printf("%2d",p[i].ttime);
  }
```

Output

```
Enter no. of process : 4
Burst time for process P1 (in ms) : 10
 Burst time for process P2 (in ms) : 4
Burst time for process P3 (in ms) : 11
Burst time for process P4 (in ms) :  6

             FCFS Scheduling

----------------------------
Process B-Time T-Time  W-Time
----------------------------
   P1        10       10        0
   P2         4       14       10
   P3        11       25       14
   P4         6       31       25
----------------------------

Average waiting time          : 12.25ms
Average turn around time :  20.00ms


GANTT Chart
------------------------------------------------------------
|         P1       |    P2 |          P3         |      P4   |

------------------------------------------------------------

 0                  10       14                   25          31
```

ii) SJF Scheduling

Shortest Job First (SJF)

- Process that requires smallest burst time is processed first.
- SJF can be preemptive or non–preemptive
- When two processes require same amount of CPU utilization, FCFS is used to break the tie.
- Generally efficient as it results in minimal average waiting time.
- Can result in starvation, since long critical processes may not be processed.

Algorithm

1. Define an array of structure *process* with members *pid*, *btime*, *wtime* & *ttime*.
2. Get length of the ready queue, i.e., number of process (say *n*)

3. Obtain *btime* for each process.
4. *Sort* the processes according to their *btime* in ascending order.
   a. If two process have same *btime*, then FCFS is used to resolve the tie.
5. The *wtime* for first process is 0.
6. Compute *wtime* and *ttime* for each process as:
   a. $wtime_{i+1} = wtime_i + btime_i$
   b. $ttime_i = wtime_i + btime_i$
7. Compute average waiting time *awat* and average turn around time *atur*.
8. Display *btime*, *ttime* and *wtime* for each process.
9. Display GANTT chart for the above scheduling
10. Display *awat* and *atur*
11. Stop

Program

```
/* SJF Scheduling – sjf.c */

#include <stdio.h>

struct process
{
int pid; int
btime; int
wtime; int ttime;
} p[10], temp;

main()
{
int i,j,k,n,ttur,twat; float
awat,atur;
printf("Enter no. of process : "); scanf("%d",
&n);
for(i=0; i<n; i++)
{
printf("Burst time for process P%d (in ms) : ",(i+1)); scanf("%d",
&p[i].btime);
p[i].pid = i+1;
}
```

```
for(i=0; i<n-1; i++)
{
    for(j=i+1; j<n; j++)
    {
        if((p[i].btime > p[j].btime) ||(p[i].btime == p[j].btime && p[i].pid
        >  p[j].pid))
        {
            temp = p[i];
            p[i] = p[j];
            p[j] = temp;
        }


    }
}
p[0].wtime = 0;
for(i=0; i<n;  i++)
{
    p[i+1].wtime = p[i].wtime + p[i].btime;
    p[i].ttime = p[i].wtime +  p[i].btime;
}
ttur = twat = 0;
for(i=0; i<n; i++)
{
    ttur += p[i].ttime;
    twat +=  p[i].wtime;
}
awat = (float)twat / n;
atur = (float)ttur /  n;

printf("\n SJF Scheduling\n\n"); for(i=0;
i<28; i++)
printf("-");
printf("\nProcess B-Time T-Time W-Time\n");
for(i=0; i<28; i++)
    printf("-"); for(i=0; i<n;
i++)
    printf("\nP%4d\t%4d\t%3d\t%2d",p[i].pid,p[i].btime,p[i].ttime,p[i]
                .wtie);
   printf("\n");
   for(i=0; i<28;  i++)
         printf("-");
```

```
                printf("\n\nAverage waiting time: %5.2fms", awat);
                printf("\nAverage turn around time : %5.2fms\n", atur);
                printf("\n\nGANTT
                Chart\n"); printf("-");
                for(i=0; i<(p[n-1].ttime + 2*n); i++)
                   printf("-");

                   printf("\n|");
               for(i=0; i<n; i++)
          {
          k = p[i].btime/2;
           for(j=0; j<k; j++)
               printf(" "); printf("P%d",p[i].pid);
          for(j=k+1; j<p[i].btime; j++)
               printf(" ");
          printf("|");
     }
     printf("\n-");
     for(i=0; i<(p[n-1].ttime + 2*n); i++)
     printf("-");
     printf("\n0"); for(i=0;
     i<n; i++)
     {
          for(j=0; j<p[i].btime; j++)
               printf(" ");
          printf("%2d",p[i].ttime);
     }
     }
```

Output

```
Enter no. of process :       5
Burst time for process       P1 (in ms) :     10
Burst time for process       P2 (in ms) :     6
Burst time for process       P3 (in ms) :     5
Burst time for process       P4 (in ms) :     6
Burst time for process       P5 (in ms) :     9
SJF Scheduling

----------------------------
Process B-Time T-Time  W-Time
----------------------------
   P3     5       5       0
   P2     6       11      5
   P4     6       17      11
   P5     9          26        17
   P1     10         36        26
----------------------------

Average waiting time          : 11.80ms
Average turn around time :  19.00ms



GANTT Chart
----------------------------------------------
|   P3   |     P2   |     P4   |       P5       |        P1       |

 0       5         11          17              26               36
```

iii) Priority Scheduling

Priority

- Process that has higher priority is processed first.
- Prioirty can be preemptive or non–preemptive
- When two processes have same priority, FCFS is used to break the tie.
- Can result in starvation, since low priority processes may not be processed.

Algorithm

1. Define an array of structure *process* with members *pid*, *btime*, *pri*, *wtime* & *ttime*.
2. Get length of the ready queue, i.e., number of process (say *n*)
3. Obtain *btime* and *pri* for each process.
4. *Sort* the processes according to their *pri* in ascending order.
   a. If two process have same *pri*, then FCFS is used to resolve the tie.
5. The *wtime* for first process is 0.
6. Compute *wtime* and *ttime* for each process as:
   a. $wtime_{i+1} = wtime_i + btime_i$
   b. $ttime_i = wtime_i + btime_i$
7. Compute average waiting time *awat* and average turn around time *atur*

8. Display the *btime*, *pri*, *ttime* and *wtime* for each process.
9. Display GANTT chart for the above scheduling
10. Display *awat* and *atur*
11. Stop

Program

```
/* Priority Scheduling     - pri.c */

#include <stdio.h>

struct process
{
int pid; int
btime; int
pri; int
wtime; int
ttime;
} p[10], temp;

main()
{
int i,j,k,n,ttur,twat;
float awat,atur;

printf("Enter no. of process : ");
scanf("%d", &n);
for(i=0; i<n; i++)
{
printf("Burst time for process P%d (in ms) : ", (i+1));
scanf("%d", &p[i].btime);
printf("Priority for process P%d : ", (i+1));
scanf("%d", &p[i].pri);
p[i].pid = i+1;
}


for(i=0; i<n-1; i++)
{
for(j=i+1; j<n; j++)
{
if((p[i].pri > p[j].pri) ||
(p[i].pri == p[j].pri && p[i].pid > p[j].pid))
{
```

```
temp = p[i];
p[i] = p[j];
p[j] = temp;
}
}
}
p[0].wtime = 0;
for(i=0; i<n;  i++)
{
p[i+1].wtime = p[i].wtime + p[i].btime;
p[i].ttime = p[i].wtime +  p[i].btime;
}
ttur = twat = 0;
for(i=0; i<n;  i++)
{
ttur += p[i].ttime;
twat +=  p[i].wtime;
}
awat = (float)twat / n;
atur = (float)ttur /  n;

printf("\n\t Priority Scheduling\n\n");
for(i=0; i<38; i++)
printf("-");
printf("\nProcess B-Time  Priority  T-Time W-Time\n");
for(i=0; i<38; i++)
printf("-");
 for (i=0; i<n;  i++)
printf("\nP%4d\t%4d\t%3d\t%4d\t%4d",p[i].pid,p[i].btime,p[i].pri,p[i].tti
me,p[i].wtime);
printf("\n");
for(i=0; i<38;  i++)
printf("-");

printf("\n\nAverage  waiting  time: %5.2fms", awat);
printf("\nAverage turn around time : %5.2fms\n",  atur);
printf("\n\nGANTT Chart\n");
printf("-");
for(i=0; i<(p[n-1].ttime + 2*n); i++)
printf("-");
printf("\n|");
for(i=0; i<n;  i++)
```

```
{
k = p[i].btime/2;
for(j=0; j<k;  j++)
printf(" ");
printf("P%d",p[i].pid);
for(j=k+1; j<p[i].btime; j++)
printf(" ");
printf("|");
}
printf("\n-");
for(i=0; i<(p[n-1].ttime + 2*n); i++)
printf("-");
printf("\n0");
for(i=0; i<n;  i++)
{
for(j=0; j<p[i].btime; j++)
printf(" ");
printf("%2d",p[i].ttime);
}
}
```

Output

```
Enter no. of process :  5
Burst time for process  P1        (in    ms) :  10
Priority for process P1  :        3
Burst time for process  P2        (in    ms) :  7
Priority for process P2  :        1
Burst time for process  P3        (in    ms) :  6
Priority for process P3  :        3
Burst time for process  P4        (in    ms) :  13
Priority for process P4  :        4
Burst time for process  P5        (in    ms) :  5
Priority for process P5  :        2


                   Priority Scheduling

    ----------------------------------------------------------
    Process B-Time  Priority  T-Time      W-Time
    ------------------------------------------------------------
       P2        7        1         7         0
       P5        5        2        12         7
       P1       10        3        22        12
       P3        6        3        28        22
       P4       13        4        41        28
```

```
--------------------------------------------------------

Average waiting time          : 13.80ms
Average turn around time :  22.00ms

GANTT Chart
----------------------------------------------------------------------
|    P2    |   P5   |        P1       |     P3   |          P4          |

----------------------------------------------------------------------

0           7       12            22        28                   41
```

iv) Round Robin

- All processes are processed one by one as they have arrived, but in rounds.
- Each process cannot take more than the time slice per round.
- Round robin is a fair preemptive scheduling algorithm.
- A process that is yet to complete in a round is preempted after the time slice and put at the end of the queue.
- When a process is completely processed, it is removed from the queue.

Algorithm

1. Get length of the ready queue, i.e., number of process (say *n*)
2. Obtain *Burst* time B_i for each processes P_i.
3. Get the *time slice* per round, say *TS*
4. Determine the number of rounds for each process.
5. The wait time for first process is 0.
6. If $B_i > TS$ then process takes more than one round. Therefore turnaround and waiting time should include the time spent for other remaining processes in the same round.
7. Calculate *average* waiting time and turn around time
8. Display the GANTT chart that includes
   a. order in which the processes were processed in progression of rounds
   b. Turnaround time T_i for each process in progression of rounds.
9. Display the *burst* time, *turnaround* time and *wait* time for each process (in order of rounds they were processed).
10. Display *average* wait time and turnaround time
11. Stop

Program

```
/* Round robin scheduling       - rr.c */

 #include <stdio.h>

main()
{
     int i,x=-1,k[10],m=0,n,t,s=0;
     int a[50],temp,b[50],p[10],bur[10],bur1[10]; int
     wat[10],tur[10],ttur=0,twat=0,j=0;
     float awat,atur;

     printf("Enter no. of process : ");
     scanf("%d", &n);
     for(i=0; i<n; i++)
     {
          printf("Burst time for process P%d : ", (i+1));
          scanf("%d", &bur[i]);
          bur1[i] = bur[i];
     }
     printf("Enter the time slice (in ms) : ");
     scanf("%d", &t);

     for(i=0; i<n; i++)
     {
          b[i] = bur[i] / t;
          if((bur[i]%t) != 0)
               b[i] += 1;
          m += b[i];
     }


     printf("\n\t\tRound Robin Scheduling\n");

     printf("\nGANTT Chart\n");
      for(i=0; i<m; i++)
          printf(" ------------- ");
     printf("\n");

     a[0] = 0;
     while(j < m)
     {
      if(x == n-1)
       x = 0;
      else
            x++;
      if(bur[x] >= t)
      {
            bur[x] -= t;
            a[j+1] = a[j] + t;


            if(b[x] == 1)
            {
                  p[s] = x;
                  k[s] = a[j+1];
                  s++;
            }
            j++;
            b[x] -= 1;
            printf("P%d|", x+1);
      }
      else if(bur[x] != 0)
      {
            a[j+1] = a[j] + bur[x];
            bur[x] = 0;
            if(b[x] == 1)
            {
                  p[s] = x;
                  k[s] = a[j+1];
                  s++;
            }
            j++;
            b[x] -= 1;
            printf("P%d|",x+1);
      }
}
printf("\n");
for(i=0;i<m;i++)
      printf(" ------------- ");
printf("\n");

for(j=0; j<=m; j++)
printf("%d\t",  a[j]);
```

```
for(i=0; i<n; i++)
{
     for(j=i+1; j<n; j++)
     {
          if(p[i] >  p[j])
          {
               temp = p[i];
               p[i] = p[j];
               p[j] =  temp;

               temp = k[i];
               k[i] = k[j];
               k[j] =  temp;
          }
     }
}


for(i=0; i<n; i++)
{
     wat[i] = k[i] - bur1[i];
     tur[i] = k[i];
}
for(i=0; i<n;  i++)
{
     ttur += tur[i];
     twat +=  wat[i];
}

printf("\n\n");
for(i=0; i<30;  i++)
     printf("-");
     printf("\nProcess\tBurst\tTrnd\tWait\n");
     for(i=0; i<30; i++)
     printf("-");
     for (i=0; i<n;  i++)
     printf("\nP%-4d\t%4d\t%4d\t%4d", p[i]+1, bur1[i], tur[i],wat[i]);
printf("\n");
for(i=0; i<30;  i++)
     printf("-");
```

```
awat = (float)twat / n;
atur = (float)ttur /  n;
       printf("\n\nAverage waiting time: %.2f ms", awat);
       printf("\nAverage turn around time : %.2f ms\n", atur);
  }
```

Output

```
Enter no. of process : 5
Burst time for process P1 : 10 Burst
time for process P2 : 29 Burst time for
process P3 : 3 Burst time for process
P4 : 7 Burst time for process P5 : 12
Enter the time slice (in ms) :  10

                    Round Robin Scheduling

GANTT Chart
-------------------------------------------------------------------------------------------
--------
P1     |   P2    |   P3  |   P4  |   P5  |   P2    |   P5    |     P2  |
_______________________________________________________________________________
0       10         20        23         30         40         50         52         61

-------------------------------------------
Process  Burst       Trnd       Wait
----------------------------------------------
P1            10          10          0
P2            29          61         32
P3             3          23         20
P4             7          30         23
P5            12          52         40
--------------------------------------------

Average waiting time          : 23.00 ms
Average turn around time : 35.20  ms
```

Result

Thus waiting time and turnaround time for processes based on CPU Scheduling algorithms has been executed.

Exp.No : 5	**Illustrate the Interprocess Communication Strategy**
Date:	

Aim:

To implement the interprocess communication using shared memory.

ALGORITHM:

1. Start the program
2. Declare the necessary variables
3. shmat() and shmdt() are used to attach and detach shared memory segments. They are prototypes as follows: void *shmat(int shmid, const void *shmaddr, int shmflg); int shmdt(const void *shmaddr);
4. shmat() returns a pointer, shmaddr, to the head of the shared segment associated with a valid shmid. shmdt() detaches the shared memory segment located at the address indicated by shmaddr
5. Shared1.c simply creates the string and shared memory portion.
6. Shared2.c attaches itself to the created shared memory portion and uses the string (printf)
7. Stop the program.

PROGRAM:

Shared1.c:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#define SHMSZ 27
main()
{
char c;
 int shmid;
key_t key; char *shm, *s; key = 5678;
if ((shmid = shmget(key, SHMSZ, IPC_CREAT | 0666)) < 0)
{
perror("shmget");
exit(1);
}
```

```
        if ((shm = shmat(shmid, NULL, 0)) == (char *) -1)
        {
        perror("shmat");
        exit(1);
}
```

Shared2.c

```
#include <sys/ipc.h>
#include <sys/shm.h>
 #include <stdio.h>
#define SHMSZ 27
main()
{
int shmid;
 key_t key;
char *shm, *s;
key = 5678;
if ((shmid = shmget(key, SHMSZ, 0666)) < 0)
{
perror("shmget");
exit(1);
}
if ((shm = shmat(shmid, NULL, 0)) == (char *) -1)
 {
 Perror("shmat");
exit(1);
}
 for (s = shm; *s != NULL; s++)
 putchar(*s);
putchar('\n');
*shm = '*';
exit(0);
}
```

OUTPUT:

```
Abcdefghijklmnopqrstuvwxyz
```

Result

Thus the interprocess communication strategy using shared memory has been executed.

Exp.No : 6	**Implement mutual exclusion by Semaphores**
Date:	

Aim:

To demonstrate the utility of semaphore in mutual exclusion.

Semaphore

- The POSIX system in Linux has its own built-in semaphore library.
- To use it, include semaphore.h.
- Compile the code by linking with -lpthread -lrt.
- To lock a semaphore or wait, use the **sem_wait** function.
- To release or signal a semaphore, use the **sem_post** function.
- A semaphore is initialised by using **sem_init**(for processes or threads)
- To declare a semaphore, the data type is sem_t.

Algorithm

1. 2 threads are being created, one 2 seconds after the first one.
2. But the first thread will sleep for 4 seconds after acquiring the lock.
3. Thus the second thread will not enter immediately after it is called, it will enter 4 – 2= 2 secs after it is called.
4. Stop.

Program

```
/* C program to demonstrate working of Semaphores */

#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

sem_t mutex;

void* thread(void* arg)
{
        //wait  sem_wait(&mutex);
        printf("\nEntered..\n");

        //critical section
        sleep(4);
        //signal
        printf("\nJust Exiting...\n");
        sem_post(&mutex);
    }

    int main()
    {
        sem_init(&mutex, 0, 1); pthread_t
        t1,t2;
        pthread_create(&t1,NULL,thread,NULL);
        sleep(2);
        pthread_create(&t2,NULL,thread,NULL);
        pthread_join(t1,NULL);
        pthread_join(t2,NULL);
        sem_destroy(&mutex);
        return 0;
    }
```

Output

```
$ gcc sem.c -lpthread

$ ./a.out Entered..
Just Exiting... Entered..
Just Exiting...
```

Result

Thus semaphore implementation has been demonstrated.

Exp.No : 7	**Bankers Algorithm for Deadlock Avoidance**
Date:	

Aim:

To write a C program to avoid Deadlock using Banker's Algorithm.

ALGORITHM

1. Start the program.
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.
5. Check whether it's possible to allocate.
6. If it is possible then the system is in safe state.
7. Else system is not in safety state.
8. If the new request comes then check that the system is in safety or not if we allow the request.
9. Stop.

PROGRAM

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

main()
{
int r[1][10], av[1][10];
int all[10][10], max[10][10], ne[10][10], w[10],safe[10];
int i=0, j=0, k=0, l=0, np=0, nr=0, count=0,   cnt=0;
clrscr();
printf("enter the number of processes in a system");
scanf("%d", &np);
printf("enter the number of resources in a system");
scanf("%d",&nr);
for(i=1; i<=nr; i++)
{
printf("Enter no. of instances of resource R%d " ,i);
scanf("%d", &r[0][i]);
av[0][i] = r[0][i];
}
 for(i=1; i<=np; i++)
for(j=1; j<=nr;  j++)
          all[i][j] = ne[i][j] = max[i][j] =  w[i]=0;
          printf("Enter the allocation matrix");
          for(i=1; i<=np; i++)
        {
           for(j=1; j<=nr; j++)
           {
            scanf("%d", &all[i][j]);
            av[0][j] = av[0][j] - all[i][j];
           }
         }
printf("Enter the maximum matrix");
for(i=1; i<=np; i++)
{
     for(j=1; j<=nr; j++)
     {
               scanf("%d",&max[i][j]);
     }
}

for(i=1; i<=np; i++)
{
     for(j=1; j<=nr; j++)
     {
          ne[i][j] = max[i][j] - all[i][j];
     }
}

for(i=1; i<=np; i++)
{
     printf("pocess P%d", i); for(j=1;
     j<=nr; j++)
     {
          printf("\n allocated %d\t",all[i][j]);
          printf("maximum %d\t",max[i][j]);
          printf("need %d\t",ne[i][j]);
     }
     printf("\n_____________________________\n");
  }
printf("\nAvailability "); for(i=1;
    i<=nr; i++)
         printf("R%d %d\t", i, av[0][i]);
    printf("\n________________"); printf("\n
    safe sequence");


    for(count=1; count<=np; count++)
    {
       for(i=1; i<=np; i++)
       {
             Cnt = 0;
             for(j=1; j<=nr; j++)
             {
                  if(ne[i][j] <= av[0][j] && w[i]==0)
                  cnt++;
             }
             if(cnt == nr)
             {
                  k++;
                  safe[k] = i;
                  for(l=1; l<=nr;  l++)
                       av[0][l] = av[0][l] + all[i][l];
                       printf("\n P%d ",safe[k]); printf("\t
                  Availability ");
                       for(l=1; l<=nr; l++)
                       printf("R%d %d\t", l, av[0][l]);
                       w[i]=1;
             }
        }
    }
    getch();
}
```

Output

```
enter the number of processes in a system 3
enter the number of resources in a system   3

enter no. of instances of resource R1 10
enter no. of instances of resource R2  7
enter no. of instances of resource R3  7

Enter the allocation matrix 3 2 1
```

```
1 1 2
4 1 2

Enter the maximum matrix 4
4 4
3 4 5
5 2 4


Process P1
  allocated 3        maximum 4        need 1
  allocated 2        maximum 4        need 2
  allocated 1        maximum 4        need 3
____________________________________________
Process  P2
  allocated 1       maximum  3        need 2
  allocated 1       maximum  4        need 3
  allocated 2       maximum  5        need 3

____________________________________________



Process  P3
  allocated 4       maximum  5        need 1
  allocated 1       maximum  2        need 1
  allocated 2       maximum  4        need 2
______________________________________________


Availability     R1 2                 R2 3      R3 2
____________________________________________

  safe sequence

  P3          Availability R1 6        R2 4     R3 4
  P1          Availability R1 9        R2 6     R3 5
  P2          Availability R1 10       R2 7     R3 7
```

Result

Thus banker's algorithm for dead lock avoidance was executed successfully.

Exp.No : 8	**Deadlock Detection**
Date:	

Aim

To write a C program to implement Deadlock Detection Algorithm.

Algorithm

1. Mark each process that has a row in the Allocation matrix of all zeros.
2. Initialize a temporary vector to equal the Available vector.
3. Find an indexing such that processing is currently unmarked and their th row of Q
4. It is less than or equal to W. That is, Q ik … Wk, for 1 … k … m . If no such row is found, terminate the algorithm.
5. If such a row is found, mark processing and add the corresponding row of the allocation matrix to W . That is, set Wk = Wk + Aik, for 1 … k … m
6. Return to step 3.

Program

```
#include<stdio.h>
#include<conio.h> int
max[100][100]; int
alloc[100][100]; int
need[100][100]; int
avail[100];
int n, r; void
input(); void
show(); void
cal();

main()
{
int i,j;
printf("Deadlock Detection Algo\n"); input();
show();
cal();
getch();
}
```

```
void input()
{
int i,j;
printf("Enter the no of Processes\t");
scanf("%d",&n);
printf("Enter the no of resource instances\t");
scanf("%d", &r);
printf("Enter the Max Matrix\n");
for(i=0; i<n; i++)
for(j=0; j<r;  j++)
scanf("%d", &max[i][j]);
printf("Enter the Allocation Matrix\n");
for(i=0; i<n; i++)
for(j=0; j<r; j++)
scanf("%d",  &alloc[i][j]);
printf("Enter the available Resources\n");
for(j=0;j<r;j++)
scanf("%d",&avail[j]);
}

void show()
{
int i, j;
printf("Process\t Allocation\t Max\t Available\t");
for(i=0; i<n; i++)
{
printf("\nP%d\t", i+1);
for(j=0; j<r; j++)
{
printf("%d ", alloc[i][j]);
}
printf("\t");
for(j=0; j<r;  j++)
{
printf("%d ", max[i][j]);
}
printf("\t");
if(i == 0)
{
for(j=0; j<r; j++)
printf("%d ",  avail[j]);
```

```
}
}
}
void cal()
{
int finish[100], temp, need[100][100], flag=1, k, c1=0;
int dead[100];
int safe[100];
int i, j;
for(i=0; i<n; i++)
{
finish[i] = 0;
}
/*find need matrix */
for(i=0; i<n; i++)
{
for(j=0; j<r; j++)
{
need[i][j]= max[i][j] - alloc[i][j];
}
}


while(flag)
{
flag=0; for(i=0;i<n;i++)
{
int c=0; for(j=0;j<r;j++)
{
if((finish[i]==0) && (need[i][j] <=  avail[j]))
{
c++;
if(c == r)
{
for(k=0; k<r; k++)
{
avail[k] += alloc[i][j]; finish[i]=1;
flag=1;
}
if(finish[i] == 1)
{
i=n;
}
}
}
}
}
}

j = 0;
Flag = 0;
for(i=0; i<n; i++)
{
if(finish[i] == 0)
{
dead[j] = i; j++;
flag = 1;
}
}
if(flag == 1)
{
printf("\n\nSystem is in deadlock prevention and deadlock process are\n");
for(i=0;i<n;i++)
{
printf("P%d\t", dead[i]);
}
}
else
{
printf("\nNo Deadlock Occur");
}
}
```

Output

```
********** Deadlock Detection Algo ************

Enter the no of Processes              3
Enter the no of resource instances 3
Enter the Max Matrix
3 6 0
4 3 3
3 4 4
Enter the Allocation Matrix
```

```
3 3 3
2 0 3
1 2 4
Enter the available Resources 1 2 0

 Process   Allocation          Max      Available
 P1        3 3 3               3 6 0    1 2 0
 P2        2 0 3               4 3 3
 P3        1 2 4               3 4 4

System is in Deadlock and the Deadlock processes are P0 P1 P2
```

Result

Thus using given state of information deadlocked processes were determined.

Exp.No : 9	Threading
Date:	

Aim:

To write C program to implement Threading.

Algorithm

1. Create two threads.
2. Let the threads share a common resource, say counter.
3. Even if thread2 si scheduled to start while thread was not done, access to shared resource is not done as it is locked by mutex.
4. Once thread1 completes, thread2 starts execution.
5. Stop.

Program

```
#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
pthread_t tid[2];
int counter;
pthread_mutex_t lock;
void* trythis(void *arg)
{
  pthread_mutex_lock(&lock);
  unsigned long i = 0;
  counter += 1;
  printf("\n Job %d has started\n", counter);
  for(i=0; i<(0xFFFFFFFF);i++);
  printf("\n Job %d has finished\n", counter);
  pthread_mutex_unlock(&lock);
  return NULL;
}
main()
{
int i = 0; int error;
if (pthread_mutex_init(&lock, NULL) != 0)
{
        printf("\n mutex init has failed\n"); return 1;
    }
    while(i < 2)
    {
        err = pthread_create(&(tid[i]), NULL, &trythis, NULL);
        if (error != 0)
            printf("\nThread   can't be created:[%s]", strerror(error));
            i++;
    }

    pthread_join(tid[0], NULL);
    pthread_join(tid[1], NULL);
    pthread_mutex_destroy(&lock);

    return 0;
}
```

Output

```
$ gcc filename.c -lpthread

$ ./a.out
Job 1 started
Job 1 finished
Job 2 started
Job 2 finished
```

Result

Thus concurrent threads were synchronized using mutex lock.

Exp.No : 10	**Paging Technique of Memory Management**
Date:	

Aim:

To implement the paging Technique using C program.

Algorithm:

1. Get process size
2. Compte no. of pages available and display it
3. Get relative address
4. Determine the corresponding page
5. Display page table
6. Display the physical address.

Program

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

main()
{
int size, m, n, pgno, pagetable[3]={5,6,7}, i, j, frameno;
double m1;
int ra=0, ofs;
printf("Enter process size (in KB of max 12KB):");
scanf("%d", &size);
m1 = size / 4;
n = ceil(m1);
printf("Total No. of pages: %d", n);
printf("\nEnter relative address (in hexa) \n");
scanf("%d", &ra);
pgno = ra / 1000;
ofs = ra %  1000;
printf("page no=%d\n", pgno);
printf("page table");
for(i=0;i<n;i++)
printf("\n %d [%d]", i, pagetable[i]);
frameno = pagetable[pgno];
printf("\nPhysical address: %d%d", frameno, ofs);
}
```

Output

```
Enter process size (in KB of max 12KB):12
Total No. of pages: 3
Enter relative address (in hexa): 2643
page no=2
page table
  0 [5]
  1 [6]
  2 [7]
Physical address: 7643
```

Result

Thus physical address for the given logical address is determing using Paging technique.

Exp.No : 11	Memory Allocation Methods
Date:	

Aim:

To Write C programs to implement the following Memory Allocation Methods

a. First Fit
b. Worst Fit
c. Best Fit

Algorithm (First Fit):

- The first-fit, best-fit, or worst-fit strategy is used to select a free hole from the set of available holes.
- Allocate the first hole that is big enough.
- Searching starts from the beginning of set of holes

Steps:

1. Declare structures *hole* and *process* to hold information about set of holes and processes respectively.
2. Get number of holes, say *nh*.
3. Get the size of each hole
4. Get number of processes, say *np*.
5. Get the memory requirements for each process.
6. Allocate processes to holes, by examining each hole as follows:
   a. If hole size > process size then
      i. Mark process as allocated to that hole.
      ii. Decrement whole size by process size.
   b. Otherwise check the next from the set of hole
7. Print the list of process and their allocated holes or unallocated status.
8. Print the list of holes, their actual and current availability.
9. Stop.

Program

```
/* First fit allocation - ffit.c */
#include <stdio.h>
  struct process
  {
       int size;
```

```
    int flag;
    int  holeid;
}
p[10];
struct hole
{
    int size;
    int actual;
} h[10];
main()
{
    int i, np, nh,  j;
    printf("Enter the number of Holes : ");
    scanf("%d", &nh);
    for(i=0; i<nh; i++)
    {
        printf("Enter size for hole H%d : ",i);
        scanf("%d", &h[i].size);
        h[i].actual  =h[i].size;
    }
    printf("\nEnter number of process : " );
    scanf("%d",&np);
    for(i=0;i<np;i++)
    {
        printf("enter the size of process P%d : ",i);
        scanf("%d", &p[i].size);
        p[i].flag = 0;
    }
    for(i=0; i<np; i++)
    {
        for(j=0; j<nh; j++)
        {
            if(p[i].flag != 1)
            {
                if(p[i].size <=  h[j].size)
                {
                    p[i].flag = 1; p[i].holeid = j; h[j].size -
                    =  p[i].size;
                }
            }
        }
    }
```

```
printf("\n\tFirst fit\n");
printf("\nProcess\tPSize\tHole");
for(i=0; i<np; i++)
{
     if(p[i].flag != 1)
          printf("\nP%d\t%d\tNot allocated", i, p[i].size);
     else
          printf("\nP%d\t%d\tH%d", i, p[i].size,  p[i].holeid);
}
printf ("\n\nHole\tActual\tAvailable");
for (i=0; i<nh ;i++)
printf ("\nH%d\t%d\t%d", i, h[i].actual, h[i].size);
printf ("\n");
}
```

Output:

```
Enter the number of Holes: 5 Size
Enter the hole: 100 size for hole
Enter H1: 500 size for hole
Enter H2: 200 size for hole
Enter H3: 300 size for hole
Enter H4: 600 size for hole

Enter the number of process: 4
Enter the size of the process P0:212
Enter the size of the process P1:417
Enter the size of the process P2:112
Enter the size of the process P3:426

First fit
Process     PSize          Hole
P0          212            H1
P1          417            H4
P2          112            H1
P3          426            Not allocated
```

Hole	Actual	Available
H0	100	100
H1	500	176
H2	200	200
H3	300	300
H4	600	183

Algorithm (Best Fit):

Best Fit:

- Allocate the smallest hole that is big enough.
- The list of free holes is kept sorted according to size in ascending order.
- This strategy produces smallest leftover holes

Steps:

1. Declare structures *hole* and *process* to hold information about set of holes and processes respectively.
2. Get number of holes, say *nh*.
3. Get the size of each hole
4. Get number of processes, say *np*.
5. Get the memory requirements for each process.
6. Allocate processes to holes, by examining each hole as follows:
   a. Sort the holes according to their sizes in ascending order
   b. If hole size > process size then
      i. Mark process as allocated to that hole.
      ii. Decrement hole size by process size.
   c. Otherwise check the next from the set of sorted hole
7. Print the list of process and their allocated holes or unallocated status.
8. Print the list of holes, their actual and current availability.
9. Stop

Program

```
#include <stdio.h>

struct process
{
    int size;
    int flag;
    int holeid;
} p[10];

struct hole
{
    int hid;
    int size;
    int actual;
} h[10];
main()
{
    int i, np, nh, j;
    void bsort(struct hole[], int);
    printf("Enter the number of Holes : ");
    scanf("%d", &nh);
    for(i=0; i<nh; i++)
    {
        printf("Enter size for hole H%d : ",i);
        scanf("%d", &h[i].size);
        h[i].actual =h[i].size;
         h[i].hid = i;
    }
    printf("\nEnter number of process : " );
    scanf("%d",&np);
    for(i=0;i<np;i++)
    {
        printf("enter the size of process P%d : ",i);
        scanf("%d", &p[i].size);
        p[i].flag = 0;
    }
    for(i=0; i<np; i++)
    {
        bsort(h, nh); for(j=0; j<nh;  j++)
        {
            if(p[i].flag != 1)
            {
                if(p[i].size <=  h[j].size)
                {
                    p[i].flag = 1;
                    p[i].holeid =  h[j].hid;
                    h[j].size -=  p[i].size;
                }
            }
        }
    }
     printf("\n\tBest fit\n");
     printf("\nProcess\tPSize\tHole");
     for(i=0; i<np; i++)
     {
          if(p[i].flag != 1)
               printf("\nP%d\t%d\tNot allocated", i, p[i].size);
          else
               printf("\nP%d\t%d\tH%d", i, p[i].size,  p[i].holeid);
     }
     printf("\n\nHole\tActual\tAvailable");
     for(i=0; i<nh ;i++)
          printf("\nH%d\t%d\t%d", h[i].hid, h[i].actual, h[i].size);
          printf("\n");
}

void bsort(struct hole bh[], int n)
{
     struct hole temp;
     int i,j;
     for(i=0; i<n-1; i++)
     {
          for(j=i+1; j<n; j++)
          {
               if(bh[i].size > bh[j].size)
               {
                    temp = bh[i]; bh[i] = bh[j];
                    bh[j] = temp;
               }
          }
     }
}
```

Output

```
Enter the number of Holes : 5
Enter size for hole H0 : 100
Enter size for hole H1 : 500
Enter size for hole H2 : 200
Enter size for hole H3 : 300
Enter size for hole H4 : 600

Enter number of process :  4
enter the size of process P0 : 212
enter the size of process P1 : 417
enter the size of process P2 : 112
enter the size of process P3 :   426

              Best fit

Process      PSize      Hole
P0           212        H3
P1           417        H1
P2           112        H2
P3           426        H4

Hole         Actual     Available
H1           500        83
H3           300        88
H2           200        88
H0           100        100
H4           600        174
```

Result:

Thus the above Memory Allocation Methods (First fit,Best fit) has been successfully completed.

Exp.No : 12	**Memory Allocation Methods**
Date:	

Aim:

To write C programs to implement the various Page Replacement Algorithms

FIFO

- Page replacement is based on when the page was brought into memory.
- When a page should be replaced, the oldest one is chosen.
- Generally, implemented using a FIFO queue.
- Simple to implement, but not efficient.
- Results in more page faults.
- The page-fault may increase, even if frame size is increased (Belady's anomaly)

Algorithm

```
1. Get length of the reference string, say l.
2. Get reference string and store it in an array, say rs.
3. Get number of frames, say nf.
4. Initalize frame array upto length nf to -1.
5. Initialize position of the oldest page, say j to 0.
6. Initialize no. of page faults, say count to 0.
7. For each page in reference string in the given order, examine:
   a. Check whether page exist in the frame array
   b. If it does not exist then
      i. Replace page in position j.
      ii. Compute page replacement position as (j+1) modulus nf.
      iii. Increment count by 1.
      iv. Display pages in frame array.
8. Print count.
9. Stop
```

Program

```
#include <stdio.h>

main()
{
int i,j,l,rs[50],frame[10],nf,k,avail,count=0;
printf("Enter length of ref. string : ");
scanf("%d", &l);
printf("Enter reference string :\n");
for(i=1; i<=l; i++)
scanf("%d", &rs[i]);
printf("Enter number of frames : ");
scanf("%d", &nf);
for(i=0; i<nf; i++)
frame[i] = -1;
j = 0;
printf("\nRef.  Str.Page frames");
for(i=1; i<=l; i++)
{
printf("\n%4d\t", rs[i]);
avail = 0;
for(k=0; k<nf; k++)
if(frame[k] ==  rs[i])
avail = 1;
if(avail == 0)
{
frame[j] = rs[i];
j = (j+1) % nf; count++;
for(k=0; k<nf; k++)
printf("%4d",  frame[k]);
}
}
printf("\n\nTotal no. of page faults :  %d\n",count);
}
```

Output

```
Enter length of ref. string : 20
Enter reference string :1 2 3 4 2 1 5 6 2 1 2 3 7 6  3
Enter number of frames : 5

     Ref. str           Page frames

        1           1   -1   -1   -1   -1
        2           1    2   -1   -1   -1
        3           1    2    3   -1   -1
        4           1    2    3    4   -1
        2
        1
        5           1    2    3    4    5
        6           6    2    3    4    5
        2
        1           6    1    3    4    5
        2           6    1    2    4    5
        3           6    1    2    3    5
        7           6    1    2    3    7
        6
        3

               Total no. of page faults: 10
```

LRU

- Pages used in the recent past are used as an approximation of future usage.
- The page that has not been used for a longer period of time is replaced.
- LRU is efficient but not optimal.
- Implementation of LRU requires hardware support, such as counters/stack.

Algorithm

```
1.  Get length of the reference string, say len.
2.  Get reference string and store it in an array, say rs.
3.  Get number of frames, say nf.
4.  Create access array to store counter that indicates a measure of recent usage.
5.  Create a function arrmin that returns position of minimum of the given array.
6.  Initalize frame array upto length nf to -1.
7.  Initialize position of the page replacement, say j to 0.
8.  Initialize freq to 0 to track page frequency
9.  Initialize no. of page faults, say count to 0.
10. For each page in reference string in the given order, examine:
        a. Check whether page exist in the frame array.
        b. If page exist in memory then
```

```
            i. Store incremented freq for that page position in access array.
        c. If page does not exist in memory then
            i. Check for any empty frames.
            ii. If there is an empty frame,
                ➢ Assign that frame to the page.
                ➢ Store incremented freq for that page position in access
                ➢ array.
                ➢ Increment count.
            iii. If there is no free frame then
                ➢ Determine page to be replaced using arrmin function.
                ➢ Store incremented freq for that page position in access array.
                ➢ Increment count.
            iv. Display pages in frame array.
11. Print count.
12. Stop.
```

Program

```
/* LRU page replacement - lrupr.c */

#include <stdio.h>
int arrmin(int[], int);
main()
{
int i,j,len,rs[50],frame[10],nf,k,avail,count=0;
int access[10], freq=0, dm;
printf("Length of Reference string : ");
scanf("%d", &len);
printf("Enter reference string :\n");
for(i=1; i<=len; i++)
scanf("%d", &rs[i]);
printf("Enter no. of frames : ");
scanf("%d", &nf);
for(i=0; i<nf; i++)
frame[i] = -1;
j = 0;
printf("\nRef. Str.Page frames");
for(i=1; i<=len; i++)
{
    printf("\n%4d\t", rs[i]); avail = 0;
    for(k=0; k<nf; k++)
```

```
            {
                if(frame[k] == rs[i])
                {
                    avail = 1;
                    access[k] = ++freq; break;
                }
            }
            if(avail == 0)
            {
                dm = 0;
                for(k=0; k<nf; k++)
                {
                    if(frame[k] == -1)
                        dm = 1;
                        break;
                }
                if(dm == 1)
                {
                    frame[k] = rs[i];
                    access[k] = ++freq;
                    count++;
                }
                else
                {
                    j = arrmin(access, nf);
                    frame[j] = rs[i];
                    access[j] = ++freq;
                    count++;
                }
                for(k=0; k<nf; k++)
                printf("%4d",frame[k]);
            }
        }
        printf("\n\nTotal no. of page faults : %d\n",  count);
    }
    int arrmin(int a[], int n)
    {
           int i, min  = a[0];
          for(i=1;  i<n;  i++)
                  if (min >  a[i])

                min = a[i];
        for(i=0; i<n;  i++)
              if (min == a[i])
                 return i;
}
```

Output

```
Length of Reference string : 15
Enter reference string:
Total no. of page faults : 8
12342      15621     23763
Enter the no.of frames: 5

Ref.Str      Page frames
   1        1   -1   -1   -1   -1
   2        1    2   -1   -1   -1
   3        1    2    3   -1   -1
   4        1    2    3    4   -1
   2
   1
   5        1    2    3    4    5
   6        1    2    6    4    5
   2
   1
   2
   3        1    2    6    3    5
   7        1    2    6    3    7
   6
   3

Total no. of page faults: 8
```

Result:

Thus the above Page replacement algorithms (FIFO, LRU) have been executed successfully.

Exp.No : 13	**File Organization Techniques**
Date:	

Aim:

To write C programs to implement the various File Organization Techniques (Single Level Directory and Two Level Directory).

i) Single Level Directory:

Algorithm

```
1. Get name of directory for the user to store all the files
2. Display menu
3. Accept choice
4. If choice =1 then
        • Accept filename without any collission Store it in the directory
5. If choice =2 then
        • Accept filename
        • Remove filename from the directory array
6. If choice =3 then
        • Accept filename
        • Check for existence of file in the directory array
7. If choice =4 then
        • List all files in the directory array
8. If choice =5 then
      Stop
```

Program

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
struct
{
    char dname[10];
    char fname[25][10];
    int fcnt;
}dir;

main()
{
    int i, ch;
    char f[30];
    clrscr();
```

```
dir.fcnt = 0;
printf("\nEnter name of directory -- ");
scanf("%s", dir.dname);
while(1)
{
    printf("\n\n 1. Create File\t2. Delete File\t3. Search File \n4. Display Files\t5.
    Exit\nEnter your choice--");
    scanf("%d",&ch);
    switch(ch)
    {
        case 1:
            printf("\n Enter the name of the file -- ");
            scanf("%s",  dir.fname[dir.fcnt]);
            dir.fcnt++;
            break;

        case 2:
            printf("\n Enter the name of the file -- ");
            scanf("%s", f);
            for(i=0; i<dir.fcnt; i++)
            {
                if(strcmp(f, dir.fname[i]) == 0)
                {
                    printf("File %s is deleted ",f);
                    strcpy(dir.fname[i], dir.fname[dir.fcnt-1]);
                    break;
                }
            }
            if(I == dir.fcnt)
                printf("File %s not found", f);
            else
                dir.fcnt--;
                break;

          case 3:
            printf("\n Enter the name of the file -- ");
            scanf("%s", f);
            for(i=0; i<dir.fcnt; i++)
            {
                if(strcmp(f, dir.fname[i]) == 0)
                {

                      printf("File %s is found ", f); break;
```

```
                        }
                    }
                    if(i == dir.fcnt)
                        printf("File %s not found", f);
                        break;
                case 4:
                    if(dir.fcnt == 0)
                        printf("\n Directory Empty");
                    else
                    {
                        printf("\n The Files are -- ");
                        for(i=0; i<dir.fcnt; i++)
                            printf("\t%s", dir.fname[i]);
                    }
                    break;
                default:
                    exit(0);
            }
        }
        getch();
}
```

Output

```
Enter name of directory -- AD

 1. Create File 2. Delete File 3. Search File
 4. Display Files 5. Exit
 Enter your choice -- 1
 Enter the name of the file -- fcfs

 1. Create File 2. Delete File 3. Search File
 4. Display Files 5. Exit
 Enter your choice -- 1
 Enter the name of the file -- sjf

 1. Create File 2. Delete File 3. Search File
 4. Display Files 5. Exit

  Enter your choice -- 1
 Enter the name of the file -- God

1. Create File 2. Delete File 3. Search File
 4. Display Files 5. Exit
 Enter your choice -- 3
 Enter the name of the file -- sjf File
sjf is found

 1. Create File 2. Delete File 3. Search File
 4. Display Files 5. Exit
 Enter your choice -- 3
 Enter the name of the file -- bank
 File bank is not found

 1. Create File 2. Delete File 3. Search File
 4. Display Files 5. Exit
 Enter your choice -- 4
 The Files are -- fcfs  sjf God   bank

 1. Create File 2. Delete File   3. Search File
 4. Display Files  5. Exit
 Enter your choice -- 2
 Enter the name of the file – God
 File God is deleted
```

ii) Two-Level Directory:

Algorithm

```
1. Display menu
2. Accept choice
3. If choice =1 then
       • Accept directory name
       • Create an entry for that directory
4. If choice =2 then
       • Get directory name
       • If directory exist then accept filename without collision else report
         error
5. If choice =3 then
       ▪ Get directory name
       ▪ If directory exist then Get filename
       ▪ If file exist in that directory then delete entry else report error
6. If choice =4 then
       ▪ Get directory name
       ▪ If directory exist then Get filename
       ▪ If file exist in that directory then Display filename else report error
7. If choice =5 then
     • Display files directory-wise
8. If choice =6 then
       Stop
```

Program

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
struct
{
    char dname[10], fname[10][10];
    int fcnt;
}dir[10];

main()
{
    int i, ch, dcnt, k;
    char f[30], d[30];
    clrscr();
    dcnt=0;
    while(1)
    {
        printf("\n\n 1. Create Directory\t 2. Create File\t  3.Delete File");
        printf("\n 4. Search File \t \t 5. Display \t 6. Exit \n Enter your choice -- ");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1:
                printf("\n Enter name of directory -- ");
                scanf("%s", dir[dcnt].dname);
                dir[dcnt].fcnt = 0;
                dcnt++;
                printf("Directory created");
                break;

            case 2:
                printf("\n Enter name of the directory -- ");
                scanf("%s", d);
                for(i=0; i<dcnt; i++)
                if(strcmp(d,dir[i].dname) == 0)
                {
                    printf("Enter name of the file -- ");
                    scanf("%s", dir[i].fname[dir[i].fcnt]);
                    dir[i].fcnt++;
                    printf("File created");
                    break;
                }
                if(i == dcnt)
                    printf("Directory %s not found",d);
                    break;

         case 3:
                printf("\nEnter name of the directory -- ");
                scanf("%s", d);
                for(i=0; i<dcnt; i++)
                {
                    if(strcmp(d,dir[i].dname) == 0)
                    {
                              printf("Enter name of the file -- ");
                              scanf("%s", f);
                              for(k=0; k<dir[i].fcnt; k++)
                              {
                                        if(strcmp(f, dir[i].fname[k]) ==  0)
                                        {
                                                printf("File %s is deleted ", f);
                                                dir[i].fcnt--;
                                                strcpy(dir[i].fname[k], dir[i].fname[dir[i].fcnt]);
                                                goto jmp;
                                        }
                              }
                              printf("File %s not found",f);
                              goto jmp;
                    }
                }

                printf("Directory %s not found",d);
                jmp : break;
            case 4:
                printf("\nEnter name of the directory -- ");
                scanf("%s", d);
                for(i=0; i<dcnt; i++)
                {
                    if(strcmp(d,dir[i].dname) == 0)
                    {
                              printf("Enter the name of the file -- ");
                              scanf("%s", f);
                              for(k=0; k<dir[i].fcnt; k++)
                              {
                                        if(strcmp(f, dir[i].fname[k]) ==  0)
```

```
                                {
                                        printf("File %s is found ", f);
                                        goto jmp1;
                                }
                        }
                        printf("File %s not found", f); goto jmp1;

                    }
                }
                printf("Directory %s not found", d); jmp1: break;

        case 5:
            if(dcnt == 0)
                printf("\nNo Directory's ");
            else
            {
                printf("\nDirectory\tFiles");
                for(i=0;i<dcnt;i++)
                {
                    printf("\n%s\t\t",dir[i].dname);
                    for(k=0;k<dir[i].fcnt;k++)
                        printf("\t%s",dir[i].fname[k]);
                }
            }
            break;

        default:
            exit(0);
        }
    }
    getch();
}
```

Output

```
1. Create Directory 2. Create File  3. Delete File  4. Search File
5. Display 6. Exit
Enter your choice -- 1
Enter name of directory – AD
Directory created

1. Create Director  2. Create File  3. Delete File  4. Search File  5.Display 6. Exit
Enter your choice -- 1
Enter name of directory -- EEE
Directory created

1. Create Directory 2. Create File  3. Delete File  4. Search File
5. Display 6. Exit
Enter your choice – 2
Enter name of the directory -- ECE
Enter name of the file -- God
File created

1. Create Directory 2. Create File  3. Delete File  4. Search File
5. Display6. Exit
Enter your choice -- 2
Enter name of the directory -- AD
Enter name of the file -- Krish
File created

1. Create Directory 2. Create File  3. Delete File  4. Search File
5. Display            6. Exit
Enter your choice -- 2
Enter name of the directory -- AD
Enter name of the file -- siva
File created

1. Create Directory 2. Create File  3. Delete File  4. Search File
5. Display            6. Exit
Enter your choice -- 2
Enter name of the directory -- ECE
Enter  name of the file -- Naveeda
File created

1. Create Directory  2. Create File  3. Delete File  4. Search File
5. Display6. Exit
Enter your choice -- 5
Directory   Files
CSE              God siva
ECE              Krish Naveeda
```

```
1. Create  Directory 2. Create File  3. Delete File  4. Search File
5. Display             6. Exit
Enter your choice -- 3
Enter name of the directory -- ECE
Enter name of the file -- Naveeda
File Naveeda is deleted
```

Result:

Thus the C programs to for various File Organization Techniques (Single Level Directory and Two Level Directory) has been executed.

Exp.No : 14	File Allocation Strategies
Date:	

Aim:

Implement the following File Allocation Strategies using C programs

a. Sequential
b. Indexed
c. Linked

a) Sequential (or) Contiguous

- Each file occupies a set of contiguous block on the disk.
- The number of disk seeks required is minimal.
- The directory contains address of starting block and number of contiguous block (length) occupied.
- Supports both sequential and direct access.
- First / best fit is commonly used for selecting a hole.

Algorithm

1. Assume no. of blocks in the disk as 20 and all are free.
2. Display the status of disk blocks before allocation.
3. For each file to be allocated:
   a. Get the *filename*, *start* address and file *length*
   b. If *start* + *length* > 20, then goto step 2.
   c. Check to see whether any block in the range (start, start + length-1) is allocated. If so, then go to step 2.
   d. Allocate blocks to the file contiguously from start block to start + length – 1.
4. Display directory entries.
5. Display status of disk blocks after allocation
6. Stop

Program

```
/* Contiguous Allocation - cntalloc.c */

#include <stdio.h>
#include <string.h>
int num=0, length[10], start[10];
char fid[20][4], a[20][4];
```

```
void directory()
{
int i;
printf("\nFile Start Length\n");
for(i=0; i<num; i++)
printf("%-4s %3d %6d\n",fid[i],start[i],length[i]);
}
void display()
{
int i;
for(i=0; i<20; i++)
printf("%4d",i);
printf("\n");
for(i=0; i<20;  i++)
printf("%4s", a[i]);
}
main()
{
int i,n,k,temp,st,nb,ch,flag;
char id[4];
for(i=0; i<20; i++)
strcpy(a[i], "");
printf("Disk space before allocation:\n");
display();
do
{
printf("\nEnter File name (max 3 char) : ");
scanf("%s", id);
printf("Enter start block : ");
scanf("%d", &st);
printf("Enter no. of blocks : ");
scanf("%d", &nb);
strcpy(fid[num], id);
length[num] = nb;
flag = 0;
if((st+nb) > 20)
{
printf("Requirement exceeds range\n");
continue;
}
for(i=st; i<(st+nb); i++)
if(strcmp(a[i], "") !=  0)
flag = 1;
if(flag == 1)
{
printf("Contiguous allocation not possible.\n");
continue;
}
start[num] = st;
for(i=st; i<(st+nb);  i++)
strcpy(a[i], id);
printf("Allocation done\n");
num++;
printf("\nAny more allocation (1. yes / 2. no)? : ");
scanf("%d", &ch);
}
while (ch == 1);
printf("\n\t\t\tContiguous Allocation\n");
printf("Directory:");
directory();
printf("\nDisk space after allocation:\n");
display();
}
```

Output

```
Disk space before allocation:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19


Enter File name (max 3 char) : cp

Enter start block : 14

Enter no. of blocks : 3

Allocation done


Any more allocation (1. yes / 2. no)? : 1


Enter File name (max 3 char) : tr

Enter start block : 18
Enter no. of blocks : 3
Requirement exceeds range

Enter File name (max 3 char) : tr

Enter start block : 10

Enter no. of blocks : 3

Allocation done
Any more allocation (1. yes / 2. no)? : 1
Enter File name (max 3 char) : mv

Enter start block : 0

Enter no. of blocks : 2

Allocation done
Any more allocation (1. yes / 2. no)? : 1
Enter File name (max 3 char) : ps

Enter start block : 12
Enter no. of blocks : 3
Contiguous allocation not possible.
Any more allocation (1. yes / 2. no)? :2
                              Contiguous Allocation
Directory:
File Start Length cp
        14      3
tr      10      3
mv       0      2
```

b)Linked

- ➢ Each file is a linked list of disk blocks.
- ➢ The directory contains a pointer to first and last blocks of the file.
- ➢ The first block contains a pointer to the second one, second to third and so on.
- ➢ File size need not be known in advance, as in contiguous allocation.
- ➢ No external fragmentation.
- ➢ Supports sequential access only.

c) Indexed

- In indexed allocation, all pointers are put in a single block known as index block.
- The directory contains address of the index block.
- The i^th^ entry in the index block points to i^th^ block of the file.
- Indexed allocation supports direct access.
- It suffers from pointer overhead, i.e wastage of space in storing pointers.

Algorithm

1. Get no. of files
2. Accept filenames and no. of blocks fo each file
3. Obtrain start block for each file
4. Obtain other blocks for each file
5. Check block availability before allocation
6. If block is unavailable then report error
7. Accept file name
8. Display linked file allocation blocks for that file
9. Stop

Program

```
#include <stdio.h>
 #include <conio.h>
#include <string.h>
main()
{

    static int b[20], i, j, blocks[20][20];
    char F[20][20], S[20], ch;
    int sb[20], eb[20], x, n;
    clrscr();
    printf("\n Enter no. of Files ::");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("\n Enter file %d name ::", i+1);
        scanf("%s", &F[i]);
        printf("\n Enter No. of blocks::", i+1);
        scanf("%d",&b[i]);
    }
    for(i=0;i<n;i++)
    {
        printf("\n Enter Starting block of file%d::",i+1);
```

```
            scanf("%d", &sb[i]);
            printf("\nEnter blocks for file%d::\n", i+1);
            for(j=0; j<b[i]-1;)
            {
                printf("\n Enter the %dblock ::", j+2);
                scanf("%d", &x);
                if(b[i] != 0)
                {
                    blocks[i][j] = x;
                    j++;
                }
                else
                    printf("\n Invalid block::");
            }
        }

        printf("\nEnter the Filename :");
        scanf("%s", &S);
        for(i=0; i<n; i++)
        {
            if(strcmp(F[i],S) == 0)
            {
                printf("\nFname\tBsize\tStart\tBlocks\n");
                printf("\n ------------------------------------------------------- \n");
                printf("\n%s\t%d\t%d\t", F[i], b[i], sb[i]);
                printf("%d->",sb[i]);
                for(j=0; j<b[i];  j++)
                {
                    if(b[i] != 0)
                        printf("%d->", blocks[i][j]);
                }
            }
        }
        printf("\n---------------------------------------------------------------\n");
        getch();
    }
```

Output

```
  Enter no. of  Files ::2

  Enter file 1 name ::fcfs
  Enter No. of blocks::3
```

```
Enter file 2 name ::sjf
Enter No. of  blocks::2

Enter Starting block of file1:8
Enter blocks for file1:
Enter the 2 block :3
Enter the 3block :5

Enter Starting block of file2:2
Enter blocks for file2:
Enter the 2 block ::6

Enter the Filename ::fcfs

Fname Bsize Start Blocks
---------------------------------------------
fcfs     3         8     8->3->5
---------------------------------------------
```

Result

Thus blocks for file were allocation using linked and Contiguous allocation method completed successfully.

Exp.No : 15	**Disk Scheduling Algorithms**
Date:	

Aim:

Implement the following File Allocation Strategies using C programs.

Procedure:

i) FCFS Disk Scheduling:

FCFS is the simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue

Program:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int RQ[100],i,n,TotalHeadMoment=0,initial;
    printf("Enter the number of Requests\n");
    scanf("%d",&n);
    printf("Enter the Requests sequence\n");
    for(i=0;i<n;i++)
     scanf("%d",&RQ[i]);
    printf("Enter initial head position\n");
    scanf("%d",&initial);

    // logic for FCFS disk scheduling

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

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

}
```

Output:

```
Enter the number of Request
8
Enter the Requests Sequence
95 180 34 119 11 123 62 64
Enter initial head position
50
Total head movement is 644
```

ii) SSTF Disk Scheduling:

Shortest seek time first (SSTF) algorithm selects the disk I/O request which requires the least disk arm movement from its current position regardless of the direction. It reduces the total seek time as compared to FCFS.

Program:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int RQ[100],i,n,TotalHeadMoment=0,initial,count=0;
    printf("Enter the number of Requests\n");
    scanf("%d",&n);
    printf("Enter the Requests sequence\n");
    for(i=0;i<n;i++)
     scanf("%d",&RQ[i]);
    printf("Enter initial head position\n");
    scanf("%d",&initial);

    // logic for sstf disk scheduling

        /* loop will execute until all process is completed*/
    while(count!=n)
    {
        int min=1000,d,index;
        for(i=0;i<n;i++)
        {
           d=abs(RQ[i]-initial);
           if(min>d)
           {
               min=d;
               index=i;
           }

        }
        TotalHeadMoment=TotalHeadMoment+min;
        initial=RQ[index];
        // 1000 is for max
        // you can use any number
        RQ[index]=1000;
        count++;
    }

    printf("Total head movement is %d",TotalHeadMoment);
    return 0;
}
```

Output:

```
Enter the number of Request
8
Enter Request Sequence
95 180 34 119 11 123 62 64
Enter initial head Position
50
Total head movement is 236
```

iii) **Scan/ Elevator disk scheduling:**

It is also called as Elevator Algorithm. In this algorithm, the disk arm moves into a particular direction till the end, satisfying all the requests coming in its path, and then it turns backend moves in the reverse direction satisfying requests coming in its path.

It works in the way an elevator works, elevator moves in a direction completely till the last floor of that direction and then turns back.

Program:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
    printf("Enter the number of Requests\n");
    scanf("%d",&n);
    printf("Enter the Requests sequence\n");
    for(i=0;i<n;i++)
     scanf("%d",&RQ[i]);
    printf("Enter initial head position\n");
    scanf("%d",&initial);
```

```
printf("Enter total disk size\n");
scanf("%d",&size);
printf("Enter the head movement direction for high 1 and for low 0\n");
scanf("%d",&move);

// logic for Scan disk scheduling

   /*logic for sort the request array */
for(i=0;i<n;i++)
{
   for(j=0;j<n-i-1;j++)
   {
      if(RQ[j]>RQ[j+1])
      {
         int temp;
         temp=RQ[j];
         RQ[j]=RQ[j+1];
         RQ[j+1]=temp;
      }

   }
}

int index;
for(i=0;i<n;i++)
{
   if(initial<RQ[i])
   {
      index=i;
      break;
   }
}

// if movement is towards high value
if(move==1)
{
   for(i=index;i<n;i++)
   {
      TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
      initial=RQ[i];
   }
   //  last movement for max size
   TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
   initial = size-1;
```

```
        for(i=index-1;i>=0;i--)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];
        }
    }
    // if movement is towards low value
    else
    {
        for(i=index-1;i>=0;i--)
        {
           TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
           initial=RQ[i];
        }
        //  last movement for min size
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
        initial =0;
        for(i=index;i<n;i++)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];

        }
    }

    printf("Total head movement is %d",TotalHeadMoment);
    return 0;
}
```

Output:

```
Enter the number of Request
8
Enter the Requests Sequence
95 180 34 119 11 123 62 64
Enter initial head position
50
Enter total disk size
200
Enter the head movement direction for high 1 and for low 0
1
Total head movement is 337
```

https://www.easycodingzone.com/2021/07/c-program-of-disk-scheduling-algorithms.html