

**LAB MANUAL OF  
OPERATING SYSTEMS LAB  
(Linux Programming and Administration)  
ETCS 352**



Maharaja Agrasen Institute of Technology, PSP area,  
Sector – 22, Rohini, New Delhi – 110085  
( Affiliated to Guru Gobind Singh Indraprastha University,  
New Delhi )



# **MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY**

## **VISION**

To nurture young minds in a learning environment of high academic value and imbibe spiritual and ethical values with technological and management competence.

## **MISSION**

The Institute shall endeavor to incorporate the following basic missions in the teaching methodology:

### **Engineering Hardware – Software Symbiosis**

Practical exercises in all Engineering and Management disciplines shall be carried out by Hardware equipment as well as the related software enabling deeper understanding of basic concepts and encouraging inquisitive nature.

### **Life – Long Learning**

The Institute strives to match technological advancements and encourage students to keep updating their knowledge for enhancing their skills and inculcating their habit of continuous learning.

### **Liberalization and Globalization**

The Institute endeavors to enhance technical and management skills of students so that they are intellectually capable and competent professionals with Industrial Aptitude to face the challenges of globalization.

### **Diversification**

The Engineering, Technology and Management disciplines have diverse fields of studies with different attributes. The aim is to create a synergy of the above attributes by encouraging analytical thinking.

### **Digitization of Learning Processes**

The Institute provides seamless opportunities for innovative learning in all Engineering and Management disciplines through digitization of learning processes using analysis, synthesis, simulation, graphics, tutorials and related tools to create a platform for multi-disciplinary approach.

### **Entrepreneurship**

The Institute strives to develop potential Engineers and Managers by enhancing their skills and research capabilities so that they become successful entrepreneurs and responsible citizens.



## **MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY**

### **COMPUTER SCIENCE & ENGINEERING DEPARTMENT**

#### **VISION**

“To be centre of excellence in education, research and technology transfer in the field of computer engineering and promote entrepreneurship and ethical values.”

#### **MISSION**

“To foster an open, multidisciplinary and highly collaborative research environment to produce world-class engineers capable of providing innovative solutions to real life problems and fulfill societal needs.”

# **INDEX OF THE CONTENTS**

- 1. Introduction to the lab**
- 2. Lab Requirements (details of H/W & S/W to be used)**
- 3. List of Experiments as per GGSIPU**
- 4. List of experiments beyond the syllabus**
- 5. Format of the lab record to be prepared by the students.**
- 6. Marking scheme for the Practical Exam**
- 7. Instructions for each Lab Experiment**
- 8. Sample Viva – Questions**

# 1. Introduction to the Lab

## Lab Objective

The goal of this course is to provide an introduction to the internal operation of modern operating systems. The course will cover processes and threads, mutual exclusion, CPU scheduling, deadlock, memory management and operational commands of Linux Operating System..

## Course Outcomes

On successful completion of this Course, students should be able to:

S.NO	Course Outcomes	Experiments	BL
ETCS352.1	To Demonstrate CPU Scheduling Algorithms Of Process Management.	1,2,3,4	3
ETCS352.2	To Apply Memory Management Algorithms For Solving Page Replacement Policy	5,6	3
ETCS352.3	To Demonstrate Solutions For The Deadlock Problems	7,8	3
ETCS352.4	To Use The Linux Shell Scripting To Perform Various Operations.	Additional	3

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	2	2	2	2	1	-	3	-	-	1	-	-
CO2	2	2	2	2	1	-	3	-	-	1	-	-
CO3	2	2	2	2	1	-	3	-	-	1	-	-
CO4	2	2	2	2	1	-	3	-	-	1	-	-

An operating system (OS) is the software component of a computer system that is responsible for the management and coordination of activities and the sharing of the resources of the computer. The OS acts as a host for application programs that are run on the machine. As a host, one of the purposes of an OS is to handle the details of the operation of the hardware. This relieves application programs from having to manage these details and makes it easier to write applications. Almost all computers use an OS of some type.

OSs offer a number of services to application programs and users. Applications access these services through application programming interfaces (APIs) or system calls. By using these interfaces, the application can request a service from the OS, pass parameters, and receive the results of the operation. Users may also interact with the OS by typing commands or using a graphical user interface (GUI).

Common contemporary OSs include Microsoft Windows, Mac OS X, and Linux. Microsoft Windows has a significant majority of market share in the desktop and notebook computer markets, while the server and embedded device markets are split amongst several OSs.

## **Linux**

Linux (also known as GNU/Linux) is one of the most prominent examples of free software and open source development which means that typically all underlying source code can be freely modified, used, and redistributed by anyone. The name “Linux” comes from the Linux kernel started in 1991 by Linus Torvalds. The system’s utilities and libraries usually come from the GNU operating system (which is why it is also known as GNU/Linux).

Linux is predominantly known for its use in servers. It is also used as an operating system for a wide variety of computer hardware, including desktop computers, supercomputers, video game systems, and embedded devices such as mobile phones and routers.

Linux is a modular Unix-like OS. It derives much of its basic design from principles established in Unix during the 1970s and 1980s. Linux uses a monolithic kernel which handles process control, networking, and peripheral and file system access. The device drivers are integrated directly with the kernel. Much of Linux’s higher-level functionality is provided by separate projects which interface with the kernel. The GNU userland is an important part of most Linux systems, providing the shell and Unix tools which carry out many basic OS tasks. On top of the kernel, these tools form a Linux system with a GUI that can be used, usually running in the X Windows System (X).

Linux can be controlled by one or more of a text-based command line interface (CLI), GUI, or through controls on the device itself

## STEPS TO INSTALL FEDORA LINUX OPERATING SYSTEM

1. Boot computer with Fedora installation CD/DVD
2. Select Install or Upgrade Fedora.
3. Language Selection.
4. Select appropriate Keyboard.
5. Select Basic Storage device, if your hard drive is locally attached.
6. Storage device Warning, Click Yes to discard any data.
7. Set Hostname for your Fedora installation.
8. Click on Configure Network button if configuring network during installation.
9. Click Wired tab and click on Add button. Select Connect automatically, go to ipv4 settings tab and select Method and select Manual in drop down. Fill address box with IP Address, Netmask, Gateway and DNS.
10. Select nearest city in your Time Zone.
11. Set root password
12. Select appropriate Partition as per your requirement.
13. Verify File system partition here or you can edit filesystem if you want.
14. Format Warning, click on Format if you are OK with it.
15. If you are confirmed, then click on Write Change to Disk.
16. File system Formatting.
17. Install Boot Loader and if you want to set password for Boot Loader set it.
18. Click on Customize now and select your software's for installation and then click on Next.
19. Select optional Packages and click on Next install it.
20. Installation started, this may take several minutes as per selection of packages.
21. Packages installation is in progress.
22. Installation completed, Please remove CD/DVD and Reboot system.
23. Screen of GRUB Boot Loader, use arrow keys to select Fedora Linux to boot.
24. Post installation of Fedora, Welcome Screen.
25. License Information.
26. Create non-administrative user.
27. Set Date and Time for the system.
28. Hardware Profile Submitting to Fedora Project.
29. First Login Screen of Fedora.
30. Fedora Desktop Screen.

## Introduction to Shell Programming

Shell program is series of Linux commands. Shell script is just like batch file is MS-DOS but have more power than the MS-DOS batch file. Shell script can take input from user, file and output them on screen. Useful to create our own commands that can save our lots of time and to automate some task of day today life.

### Kernel

Kernel is heart of Linux O/S. It manages resource of Linux O/S. Kernel decides who will use this resource, for how long and when. It runs programs (or set up to execute binary files). It performs following task:-

- I/O management
- Process management
- Device management
- File management
- Memory management

### Linux Shell

Computer understand the language of 0's and 1's called binary language, In early days of computing, instruction are provided using binary language, which is difficult for all of us, to read and write. So in O/s there is special program called Shell. Shell accept commands in English and translate it into computers native binary language.

### Command shell

A program that interprets commands allows a user to execute commands by typing them manually at a terminal, or automatically in programs called shell scripts. A shell is not an operating system. It is a way to interface with the operating system and run commands.

### Process

Process is of program or task carried out by your PC. For e.g. \$ ls -lR, is command or a request to list files in a directory and all subdirectory in your current directory. It is a process. A process is program (command given by user) to perform some Job. In Linux when you start process, it gives a number (called PID or process-id) PID starts from 0 to 65535.

Process is required because Linux is multi-user, multitasking o/s. It means you can run more than two processes simultaneously if you wish. For e.g. To find how many files are on system for this command will be:

```
$ ls -lR | wc -l
```



This command will take lot of time to search all files on system. So such command can be run in Background or simultaneously by giving command like

```
$ ls / -R | wc -l &
```

The ampersand (&) at the end of command tells shells start command (ls / -R | wc -l) and run it in background takes next command immediately. An instance of running command is called process and the number printed by shell is called process-id (PID), this PID can be use to refer specific running process.

## Variables in Shell

To process our data/information, data must be kept in computers RAM memory. RAM memory is divided into small locations, and each location had unique number called memory location/address, which is used to hold our data. Programmer can give a unique name to this memory location/address called memory variable or variable (Its a named storage location that may take different values, but only one at a time).

In Linux (Shell), there are two types of variable:

- (1) System variables - Created and maintained by Linux itself. This type of variable defined in CAPITAL LETTERS.
- (2) User defined variables (UDV) - Created and maintained by user. This type of variable defined in lower letters.

You can see system variables by giving command like \$ set, some of the important System variables are:

System Variable	Meaning
BASH=/bin/bash	Our shell name
BASH_VERSION=1.14.7(1)	Our shell version name
COLUMNS=80	No. of columns for our screen
HOME=/home/vivek	Our home directory
LINES=25	No. of columns for our screen
LOGNAME=students	students Our logging name
OSTYPE=Linux	Our Os type
PATH=/usr/bin:/sbin:/bin:/usr/sbin	Our path settings
PS1=[\u@\h \W]\\$	Our prompt settings

PWD=/home/students/Common	Our current working directory
SHELL=/bin/bash	Our shell name
USERNAME=vivek	User name who is currently login to this PC

To define User Define Variable Syntax:

variable name=value

'value' is assigned to given 'variable name' and Value must be on right side = sign.

Example:

\$ no=10# this is ok

Rules for Naming variable name (Both UDV and System Variable)

- (1) Variable name must begin with Alphanumeric character or underscore character (\_), followed by one or more Alphanumeric character.
- (2) Don't put spaces on either side of the equal sign when assigning value to variable.
- (3) Variables are case-sensitive, just like filename in Linux.
- (4) You can define NULL variable
- (5) Do not use ?,\* etc, to name your variable names.

## Quotes

There are three types of quotes

Quotes	Name	Meaning
"	Double Quotes	"Double Quotes" - Anything enclosed in double quotes removed meaning of that characters (except \ and \$).
'	Single quotes	'Single quotes' - Enclosed in single quotes remains unchanged.
`	Back quote	`Back quote` - To execute command

## File permission and access modes

Permissions specify what a particular person may or may not do with respect to a file or directory. Permissions are important in creating a secure environment. For example you don't want other people to change your files and you also want system files to be safe from damage (either accidental or deliberate).

Linux permissions allow 3 things to do with a file or a directory, read, write and execute. They are referred to in Linux by a single letter each.

- **r** read - you may view the contents of the file.
- **w** write - you may change the contents of the file.
- **x** execute - you may execute or run the file if it is a program or script.

For every file we define 3 sets of people for whom we may specify permissions.

- **owner** - a single person who owns the file. (typically the person who created the file but ownership may be granted to someone else by certain users) □ **group** - every file belongs to a single group.
- **others** - everyone else who is not in the group or the owner.

## View Permissions

To view permissions for a file we use the long listing option for the command ls:

Syntax: `ls -l [path]`

```
Output          ls          -l
/home/ryan/linuxtutorialwork/frog.png
-rwxr---x 1 harry users 2.7K Jan 4 07:32 /home/ryan/linuxtutorialwork/frog.png
```

In the above example the first 10 characters of the output are.

- The first character identifies the file type. If it is a dash ( - ) then it is a normal file. If it is a d then it is a directory.
- The following 3 characters represent the permissions for the owner. A letter represents the presence of a permission and a dash ( - ) represents the absence of a permission. In this example the owner has all permissions (read, write and execute).
- The following 3 characters represent the permissions for the group. In this example the group has the ability to read but not write or execute. Note that the order of permissions is always read, then write then execute.
- Finally the last 3 characters represent the permissions for others (or everyone else). In this example they have the execute permission and nothing else.

## Change permissions

To change permissions on a file or directory we use a command called **chmod**. It stands for change file mode bits which is a bit of a mouthfull but think of the mode bits as the permission indicators

### Syntax: **chmod [permissions] [path]**

chmod has permission arguments that are made up of 3 components

- Who are we changing the permission for? [ugo] - user (or owner), group, others, all
- Are we granting or revoking the permission - indicated with either a plus ( + ) or minus ( - )
- Which permission are we setting? - read ( r ), write ( w ) or execute ( x ) The following examples will make their usage clearer.

Grant the execute permission to the group. Then remove the write permission for the owner.

```
ls -l frog.png
-rwxr---x 1 harry users 2.7K Jan 4 07:32 frog.png
chmod g+x frog.png ls -l frog.png
-rwxr-x--x 1 harry users 2.7K Jan 4 07:32
frog.png chmod u-w frog.png ls -l frog.png
-r-xr-x--x 1 harry users 2.7K Jan 4 07:32 frog.png We
```

can assign multiple permissions at once.

```
ls -l frog.png
-rwxr---x 1 harry users 2.7K Jan 4 07:32
frog.png chmod g+wx frog.png ls -l frog.png
-rwxrwx--x 1 harry users 2.7K Jan 4 07:32
frog.png chmod go-x frog.png ls -l frog.png
-rwxrw---- 1 harry users 2.7K Jan 4 07:32 frog.png
```

## Setting permissions shorthand

Our typical number system is decimal. It is a base 10 number system and as such has 10 symbols (0 - 9) used. Another number system is octal which is base 8 (0-7). Now it just so happens that with 3 permissions and each being on or off, we have 8 possible combinations ( $2^3$ ). The mapping of octal to binary is in the table below.

Octal	Binary
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

All 8 octal values can be represented with 3 binary bits and that every possible combination of 1 and 0 is included in it.

Three numbers and we can specify permissions for the user, group and others.

Examples.

```
ls -l frog.png
-rw-r---x 1 harry users 2.7K Jan 4 07:32 frog.png
chmod 751 frog.png ls -l frog.png
-rwxr-x--x 1 harry users 2.7K Jan 4 07:32 frog.png
chmod 240 frog.png ls -l frog.png
--w-r----- 1 harry users 2.7K Jan 4 07:32 frog.png
```

### Writing shell script

Following steps are required to write shell script:

Use any editor like vi or mcedit to write shell script. **VI**

### editor

**vi** editor can be used to edit an existing file or to create a new file from scratch. You can also use this editor to just read a text file.

The following table lists out the basic commands to use the vi editor –

1. **vi filename-** Creates a new file if it already does not exist, otherwise opens an existing file.

2. vi -R filename-Opens an existing file in the read-only mode.
3. view filename-Opens an existing file in the read-only mode.

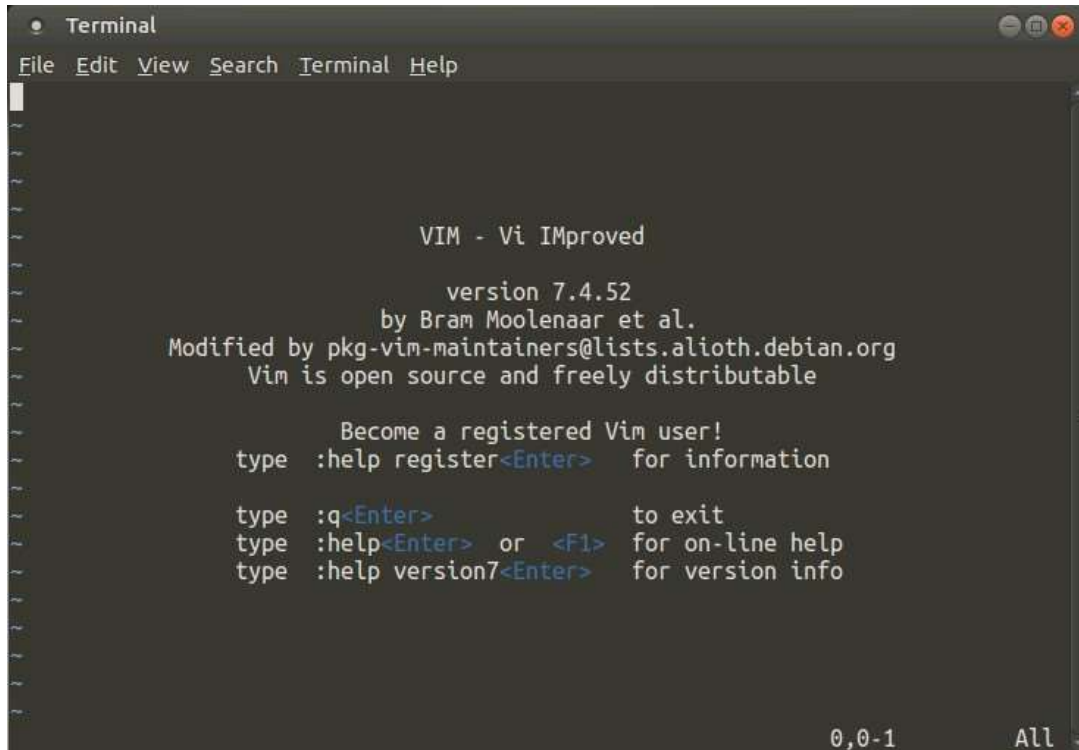
### Common vi editor command list

Purpose	Use this vi Command Syntax
To insert new text	esc + i ( You have to press 'escape' key then 'i')
To save file	esc + : + w (Press 'escape' key then 'colon' and finally 'w')
To save file with file name (save as)	esc + : + w "filename"
To quit the vi editor	esc + : + q
To quit without saving	esc + : + q!
To save and quit vi editor	esc + : + wq
To search for specified word in forward direction	esc + /word (Press 'escape' key, type /wordto-find, for e.g. to find word ' <b>shri</b> ', type as / <b>shri</b> )
To continue with search	n
To search for specified word in backward direction	esc + ?word (Press 'escape' key, type wordto-find)
To copy the line where cursor is located	esc + yy
To paste the text just deleted or copied at the cursor	esc + p
To delete entire line where cursor is located	esc + dd
To delete word from cursor position	esc + dw
To Find all occurrence of given word and Replace then globally without confirmation	esc + :\$s/word-to-find/word-to-replace/g For. e.g. :\$s/mumbai/pune/g
	Here word "mumbai" is replace with "pune"
To Find all occurrence of given word and Replace then globally with confirmation	esc + :\$s/word-to-find/word-to-replace/cg
To run shell command like ls, cp or date etc within vi	esc + :!shell-command For e.g. :!pwd

Following is an example to create a new file **testfile** if it already does not exist in the current working directory –

\$vi testfile

The above command will generate the following output –



```

VIM - Vi IMproved
          version 7.4.52
    by Bram Moolenaar et al.
Modified by pkg-vim-maintainers@lists.alioth.debian.org
Vim is open source and freely distributable

  Become a registered Vim user!
type  :help register<Enter>  for information

type  :q<Enter>              to exit
type  :help<Enter> or <F1>   for on-line help
type  :help version7<Enter> for version info

                                0,0-1      All

```

A tilde represents an unused line. If a line does not begin with a tilde and appears to be blank, there is a space, tab, newline, or some other non-viewable character present.

## Operation Modes

While working with the vi editor, two modes are here:

- **Command mode** – This mode enables you to perform administrative tasks such as saving the files, executing the commands, moving the cursor, cutting (yanking) and pasting the lines or words, as well as finding and replacing. In this mode, whatever you type is interpreted as a command.
- **Insert mode** – This mode enables you to insert text into the file. Everything that's typed in this mode is interpreted as input and placed in the file.

vi always starts in the **command mode**. To enter text, you must be in the insert mode for which simply type **i**. To come out of the insert mode, press the **Esc** key, which will take you back to the command mode.

\$ vi first #Start vi editor

```
# My first shell script #Comment clear
```

```
#clear the screen
```

```
echo "Knowledge is Power" #To print message or value of variables on screen, we use echo  
command, general form of echo command is as follow
```

After saving the above script, run the script as follows:

```
$ ./first
```

This will not run script since we have not set execute permission for our script first; to do this type command

```
$ chmod 755 first
```

```
$ ./first
```



## 2. LAB REQUIREMENTS

### Hardware Detail

Intel i3/C2D Processor/2 GB RAM/500GB HDD/MB/Lan Card/ Key Board/ Mouse/CD Drive/15” Color Monitor/ UPS	24 Nos
LaserJet Printer	1 No

### Software Detail

OS-Fedora 27

### **3. LIST OF EXPERIMENTS (As prescribed by G.G.S.I.P.U)**

**Paper Code: ETCS-352**

**L T/P C**

**Paper: Operating Systems (Linux Programming and Administration) Lab 0 2 1**

#### **List of Experiments:**

1. Write a program to implement CPU scheduling for first come first serve.
2. Write a program to implement CPU scheduling for shortest job first.
3. Write a program to perform priority scheduling.
4. Write a program to implement CPU scheduling for Round Robin.
5. Write a program for page replacement policy using a) LRU b) FIFO c) Optimal.
6. Write a program to implement first fit, best fit and worst fit algorithm for memory management.
7. Write a program to implement reader/writer problem using semaphore.
8. Write a program to implement Banker's algorithm for deadlock avoidance.

NOTE: - At least 8 Experiments out of the list must be done in the semester.

## 4. LIST OF EXPERIMENTS (Beyond the syllabus)

**Paper Code: ETCS-352**

**L T/P C**

**Paper: Operating Systems (Linux Programming and Administration) Lab 0 2 1**

**List of Experiments:**

1. Installation of Linux operating system.
2. Introduction of Basics Files, directories, system Commands of Linux.
3. i) Write a script to find the greatest of three numbers (numbers passed as command line parameters) ii) Write a script to check whether the given no. is even / odd iii) Write a script to check whether the given number is prime or not iv) Write a script to check whether the given input is a number or a string  
v) Write a script to compute no. of characters and words in each line of given file
4. i) Write a script to calculate the average of n numbers ii) Write a script to print the Fibonacci series up to n terms iii) Write a script to calculate the factorial of a given number iv) Write a script to calculate the sum of digits of the given number  
v) Write a script to check whether the given string is a palindrome

# **FORMAT OF THE LAB RECORD TO BE PREPARED BY THE STUDENTS**

The front page of the lab record prepared by the students should have a cover page as displayed below.

***NAME OF THE LAB***

***Paper Code***

Font should be (Size 20", italics bold, Times New Roman)

Faculty name

Student name

Roll No.:

Semester:

Font should be (12", Times Roman)



Maharaja Agrasen Institute of Technology, PSP Area,

Sector – 22, Rohini, New Delhi – 110085

Font should be (18", Times Roman)

## Index

Exp. no	Experiment Name	Date of performance	Date of checking	R1 (3)	R2 (3)	R3 (3)	R4 (3)	R5 (3)	Total Marks (15)	Signature

## 6. MARKING SCHEME FOR THE PRACTICAL EXAMS

There will be two practical exams in each semester.

- i. Internal Practical Exam
- ii. External Practical Exam

### **INTERNAL PRACTICAL EXAM**

It is taken by the respective faculty of the batch.

**MARKING SCHEME FOR THIS EXAM IS:**

Total Marks: 40

Division of 15 marks per practical is as follows:

### Department of Computer Science and Engineering Rubrics for Lab Assessment

Rubrics		0 Missing	1 Inadequate	2 Needs Improvement	3 Adequate
R1	Is able to identify the problem to be solved and define the objectives of the experiment.	No mention is made of the problem to be solved.	An attempt is made to identify the problem to be solved but it is described in a confusing manner, objectives are not relevant, objectives contain technical/ conceptual errors or objectives are not measurable.	The problem to be solved is described but there are minor omissions or vague details. Objectives are conceptually correct and measurable but may be incomplete in scope or have linguistic errors.	The problem to be solved is clearly stated. Objectives are complete, specific, concise, and measurable. They are written using correct technical terminology and are free from linguistic errors.
R2	Is able to design a reliable experiment that solves the problem.	The experiment does not solve the problem.	The experiment attempts to solve the problem but due to the nature of the design the data will not lead to a reliable solution.	The experiment attempts to solve the problem but due to the nature of the design there is a moderate chance the data will not lead to a reliable solution.	The experiment solves the problem and has a high likelihood of producing data that will lead to a reliable solution.
R3	Is able to communicate the details of an experimental procedure clearly and completely.	Diagrams are missing and/or experimental procedure is missing or extremely vague.	Diagrams are present but unclear and/or experimental procedure is present but important details are missing.	Diagrams and/or experimental procedure are present but with minor omissions or vague details.	Diagrams and/or experimental procedure are clear and complete.
R4	Is able to record and represent data in a meaningful way.	Data are either absent or incomprehensible.	Some important data are absent or incomprehensible.	All important data are present, but recorded in a way that requires some effort to comprehend.	All important data are present, organized and recorded clearly.
R5	Is able to make a judgment about the results of the experiment.	No discussion is presented about the results of the experiment.	A judgment is made about the results, but it is not reasonable or coherent.	An acceptable judgment is made about the result, but the reasoning is flawed or incomplete.	An acceptable judgment is made about the result, with clear reasoning. The effects of assumptions and experimental uncertainties are considered.

Each experiment will be evaluated out of 15 marks. At the end of the semester average of 8 best performed practical will be considered as marks out of 40.

## **EXTERNAL PRACTICAL EXAM**

It is taken by the concerned lecturer of the batch and by an external examiner. In this exam student needs to perform the experiment allotted at the time of the examination, a sheet will be given to the student in which some details asked by the examiner needs to be written and at the last viva will be taken by the external examiner.

### **MARKING SCHEME FOR THIS EXAM IS:**

Total Marks: 60

Division of 60 marks is as follows

1. Sheet filled by the student:	20
2. Viva Voice:	15
3. Experiment performance:	15
4. File submitted:	10

### **NOTE:**

- Internal marks + External marks = Total marks given to the students  
(40 marks)    (60 marks)                      (100 marks)
- Experiments given to perform can be from any section of the lab.

## 7. INSTRUCTIONS FOR EACH LAB EXPERIMENT

### Experiment 1

**Aim:** Write a program to implement CPU scheduling for first come first serve.

**Theory:** First come, first served (FCFS) is an operating system process scheduling algorithm and a network routing management mechanism that automatically executes queued requests and processes by the order of their arrival.

It uses non preemptive scheduling in which a process is automatically queued and processing occurs according to an incoming request or process order.

**Algorithm:**

- 1- Input the processes along with their burst time (bt).
- 2- Find waiting time (wt) for all processes.
- 3- As first process that comes need not to wait so waiting time for process 1 will be 0 i.e.  $wt[0] = 0$ .
- 4- Find waiting time for all other processes i.e. for process  $i \rightarrow wt[i] = bt[i-1] + wt[i-1]$ .
- 5- Find turnaround time = waiting\_time + burst\_time for all processes.
- 6- Find average waiting time = total\_waiting\_time / no\_of\_processes.
- 7- Similarly, find average turnaround time = total\_turn\_around\_time / no\_of\_processes.

**Sample Inputs and Outputs:**

```
Enter The Number of Processes To Execute:      3

Enter The Burst Time of Processes:

Process [1]:10
Process [2]:20
Process [3]:30

Process      Burst Time      Waiting Time      Turnaround Time
Process [1]      10.00      0.00      10.00
Process [2]      20.00      10.00      30.00
Process [3]      30.00      30.00      60.00

Average Waiting Time = 13.333333
Average Turnaround Time = 33.333332
```



**Viva - Questions:**

1. What is CPU utilization?
2. What are the different job scheduling in operating systems?
3. Which scheduling algorithm allocates the CPU first to the process that requests the CPU first?
4. The FCFS algorithm is particularly troublesome for \_\_\_\_\_
5. Which of the following statements are true?
  - i) Shortest remaining time first scheduling may cause starvation
  - ii) Preemptive scheduling may cause starvation
  - iii) Round robin is better than FCFS in terms of response time

## Experiment 2

**Aim:** Write a program to implement CPU scheduling for shortest job first.

**Theory:** Shortest job first (SJF) or shortest job next, is a scheduling policy that selects the waiting process with the smallest execution time to execute next. SJN is a non-preemptive algorithm.

### Algorithm:

- 1- Sort all the processes in increasing order according to burst time.
- 2- Then simply, apply FCFS.

### Sample Inputs and Outputs:

```
Enter the Total Number of Processes: 3
Enter Details of 3 Processes
Enter Arrival Time: 0
Enter Burst Time: 20
Enter Arrival Time: 3
Enter Burst Time: 10
Enter Arrival Time: 7
Enter Burst Time: 30
Average Waiting Time: 11.000000
Average Turnaround Time: 31.000000
```

**Viva - Questions:**

1. What is real difficulty with SJF in short term scheduling?
2. Preemptive Shortest Job First scheduling is sometimes called \_\_\_\_\_
3. Which scheduling algorithms give minimum average waiting time?
4. State true or false: Scheduling is allowing a job to use the processor?
5. What do you mean by Average turnaround time and average waiting time.

## Experiment 3

**Aim:** Write a program to perform priority scheduling.

**Theory:** Priority scheduling involves priority assignment to every process, and processes with higher priorities are carried out first, whereas tasks with equal priorities are carried out on a first-come-first-served (FCFS) or round robin basis

### Algorithm:

- 1- First input the processes with their burst time and priority.
- 2- Sort the processes, burst time and priority according to the priority.
- 3- Now simply apply FCFS algorithm.

### Sample code for priority scheduling:

```
void priorityScheduling(Process proc[], int n)
{
    // Sort processes by priority
    sort(proc, proc + n, comparison);
    cout<< "Order in which processes gets executed \n";
    for (int i = 0 ; i < n; i++)    cout<<proc[i].pid<<" "
;    findavgTime(proc, n);
}
```

### Sample Inputs and Outputs:

```

Enter Total Number of Processes:      3
Enter Burst Time and Priority For 3 Processes

Process[1]
Process Burst Time:      15
Process Priority:        3

Process[2]
Process Burst Time:      10
Process Priority:        2

Process[3]
Process Burst Time:      90
Process Priority:        1

Process ID      Burst Time      Waiting Time      Turnaround Time
Process[3]      90              0                 90
Process[2]      10              90                100
Process[1]      15              100               115

Average Waiting Time:  63.000000
Average Turnaround Time:  101.000000

```

**Viva - Questions:**

1. In priority scheduling algorithm, when a process arrives at the ready queue, its priority is compared with the priority of which process.
2. What is starvation?
3. What is aging?
4. What are the factors for assigning priority to the process?
5. If two processes come on same time, on what criteria will process be chosen for processing?

## Experiment 4

**Aim:** Write a program to implement CPU scheduling for Round Robin.

**Theory:** Round-robin (RR) is one of the algorithms employed by process and network schedulers in computing. As the term is generally used, time slices (also known as time quanta) are assigned to each process in equal portions and in circular order, handling all processes without priority **Algorithm:**

1. Create an array `rem_bt[]` to keep track of remaining burst time of processes. This array is initially a copy of `bt[]` (burst times array)
2. Create another array `wt[]` to store waiting times of processes. Initialize this array as 0.
3. Initialize time: `t = 0`

4. Keep traversing the all processes while all processes are not done. Do following for i'th process if it is not done yet.      If `rem_bt[i] > quantum`      `t = t + quantum`

`bt_rem[i] -= quantum;`

Else // Last cycle for this

process      `t = t + bt_rem[i];`

`wt[i] = t - bt[i]`

`bt_rem[i] = 0; // This process is over`

### Sample Inputs and Outputs:

```
Arrival Time: 0
Burst Time: 5

Enter Details of Process[2]:
Arrival Time: 2
Burst Time: 3

Enter Details of Process[3]:
Arrival Time: 3
Burst Time: 2

Enter Details of Process[4]:
Arrival Time: 5
Burst Time: 7

Enter Time Quantum: 2
```

Process ID	Burst Time	Turnaround Time	Waiting Time
Process[3]	2	3	1
Process[2]	3	9	6



**Viva - Questions:**

1. What do you mean by Time Slice?
2. Time quantum is defined in which scheduling algorithm?
3. What is the limitation of Round Robin Scheduling Algorithm?
4. What are the factors that affect Round Robin Scheduling Algorithm?
5. Round robin scheduling falls under the category of which scheduling Non preemptive scheduling or Preemptive scheduling.

## Experiment 5

**Aim:** Write a program for page replacement policy using a) LRU b) FIFO c) Optimal.

**Theory:** In a operating systems that use paging for memory management, page replacement algorithm are needed to decide which page needed to be replaced when new page comes in. Whenever a new page is referred and not present in memory, page fault occurs and Operating System replaces one of the existing pages with newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce number of page faults.

**Page Fault** – A page fault is a type of interrupt, raised by the hardware when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.

**Least Recently Used (LRU)** –

In this algorithm page will be replaced which is least recently used.

**First In First Out (FIFO)** –

This is the simplest page replacement algorithm. In this algorithm, operating system keeps track of all pages in the memory in a queue, oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

**Optimal Page replacement** –

In this algorithm, pages are replaced which are not used for the longest duration of time in the future.

**Algorithm (LRU):**

Let capacity be the number of pages that memory can hold. Let set be the current set of pages in memory.

1- Start traversing the pages.

If set holds less pages than capacity.

Insert page into the set one by one until the size of set reaches capacity or all page requests are processed.

Simultaneously maintain the recent occurred index of each page in a map called indexes.

Increment page fault

Else

If current page is present in set, do nothing.

Else

Find the page in the set that was least recently used. We find it using index array.

We basically need to replace the page with minimum index.

Replace the found page with current page.

Increment page faults.

Update index of current page.

2. Return page faults.

#### **Algorithm (FIFO):**

1- Start traversing the pages.

If set holds less pages than capacity.

Insert page into the set one by one until the size of set reaches capacity or all page requests are processed.

Simultaneously maintain the pages in the queue to perform FIFO.

Increment page fault

Else

If current page is present in set, do nothing.

Else

Remove the first page from the queue as it was the first to be entered in the memory

Replace the first page in the queue with the current page in the string.

Store current page in the queue.

Increment page faults.

2. Return page faults.

**Pseudo code (Optimal Page replacement):**

```
void optimalPage(int pg[], int pn, int fn)
{
    // Create an array for given number of
    // frames and initialize it as empty.
    vector<int> fr;

    // Traverse through page reference array
    // and check for miss and hit.    int hit = 0;
    for (int i = 0; i < pn; i++) {
        // Page found in a frame : HIT
        if (search(pg[i], fr)) {        hit++;
        continue;
        }

        // Page not found in a frame : MISS
        // If there is space available in frames.
        if        (fr.size()        <        fn)
        fr.push_back(pg[i]);

        // Find the page to be replaced.
        else {
            int j = predict(pg, fr, pn, i + 1);
            fr[j] = pg[i];
        }
    }
}
```

```

    cout << "No. of hits = " << hit << endl;    cout
<< "No. of misses = " << pn - hit << endl;
}

```

### Optimal Page replacement example

In this algorithm, pages are replaced which are not used for the longest duration of time in the future.

Let us consider page reference string 7 0 1 2 0 3 0 4 2 3 0 3 2 and 4 page slots.

Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots → 4 Page faults

0 is already there so → 0 Page fault.

when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future. → 1 Page fault.

0 is already there so → 0 Page fault..

4 will takes place of 1 → 1 Page Fault.

Now for the further page reference string → 0 Page fault because they are already available in the memory.

Example-2, Let's have a reference string: a, b, c, d, c, a, d, b, e, b, a, b, c, d and the size of the frame be 4.

Time Req		1	2	3	4	5	6	7	8	9	10	11	12	13	14
Page frames		a	b	c	d	c	a	d	b	e	b	a	b	c	d
0	a	a	a	a	a	a	a	a	a	a	e	e	e	e	d
1	b		b	b	b	b	b	b	b	b	b	a	a	a	a
2	c			c	c	c	c	c	c	c	c	c	b	b	b
3	d				d	d	d	d	d	e	d	d	d	c	c
FAULTS		x	x	x	x					x					x

There are 6 page faults using optimal algorithm.

Optimal page replacement is perfect, but not possible in practice as operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

### Sample Inputs and Outputs:

#### Example 1 (LRU):

Enter no of pages: 10

Enter the reference string: 7 5 9 4 3 7 9 6 2 1

Enter no of frames: 3

7		
7	5	
7	5	9
4	5	9
4	3	9
4	3	7
9	3	7
9	6	7
9	6	2
1	6	2

The no of page faults is 10

#### Example 2 (LRU):

Enter number of frames: 3

Enter number of pages: 6

Enter reference string: 5 7 5 6 7 3

5	-1	-1
5	7	-1
5	7	-1
5	7	6
5	7	6
3	7	6

Total Page Faults = 4

#### Example (FIFO)

ENTER THE NUMBER OF PAGES: 20

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

THE NUMBER OF FRAMES: 3

<u>ref string</u>	<u>page frames</u>		
7	7	-1	-1
0	7	0	-1
1	7	0	1
2	2	0	1
0			
3	2	3	1
0	2	3	0
4	4	3	0
2	4	2	0
3	4	2	3 0
			0 2 3
3			
2			
1	0	1	3
2	0	1	2
0			
1			
7	7	1	2
0	7	0	2
1	7	0	1

Page Fault Is 15

### Example 1 (Optimal page replacement output)

Enter number of frames: 3

Enter number of pages: 10

Enter page reference string: 2 3 4 2 1 3 7 5 4 3

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

**Example 2 (Optimal page replacement output)**

Sample Input: Number of frames,  $fn = 3$

Reference String,  $pg[] = \{7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1\};$

Sample Output: No. of hits = 11

No. of misses = 9

Sample Input: Number of frames,  $fn = 4$

Reference String,  $pg[] = \{7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2\};$

Sample Output: No. of hits = 7

No. of misses = 6



**Viva - Questions:**

1. Define Demand Paging, Page fault interrupt, and Thrashing?
2. Under what circumstances do page faults occur? Describe the actions taken by the operating system when a page fault occurs?
3. What is the cause of thrashing? How does the system detect thrashing? Once it detects thrashing, what can the system do to eliminate this problem?
4. What is a page and what is a frame. How are the two related?
5. What do you mean by Belady's anomaly
6. Compare LRU, FIFO and Optimal page replacement algorithm.

## Experiment 6

**Aim:** Write a program to implement first fit, best fit and worst fit algorithm for memory management.

**Theory:** First fit: In the first fit approach is to allocate the first free partition or hole large enough which can accommodate the process. It finishes after finding the first suitable free partition.

Best Fit: The best fit deals with allocating the smallest free partition which meets the requirement of the requesting process. This algorithm first searches the entire list of free partitions and considers the smallest hole that is adequate. It then tries to find a hole which is close to actual process size needed.

Worst fit: In worst fit approach is to locate largest available free portion so that the portion left will be big enough to be useful. It is the reverse of best fit.

### Algorithm (First fit):

- 1- Input memory blocks with size and processes with size.
- 2- Initialize all memory blocks as free.
- 3- Start by picking each process and check if it can be assigned to current block.
- 4- If size-of-process  $\leq$  size-of-block if yes then assign and check for next process.
- 5- If not then keep checking the further blocks.

### Algorithm (Best fit):

- 1- Input memory blocks and processes with sizes.
- 2- Initialize all memory blocks as free.
- 3- Start by picking each process and find the minimum block size that can be assigned to current process i.e., find  $\min(\text{blockSize}[1], \text{blockSize}[2], \dots, \text{blockSize}[n]) > \text{processSize}[\text{current}]$ , if found then assign it to the current process.
- 5- If not then leave that process and keep checking the further processes.

### Algorithm (Worst fit):

- 1- Input memory blocks and processes with sizes.

2- Initialize all memory blocks as free.

3- Start by picking each process and find the minimum block size that can be assigned to current process i.e., find  $\min(\text{blockSize}[1], \text{blockSize}[2], \dots, \text{blockSize}[n]) > \text{processSize}[\text{current}]$ , if found then assign it to the current process.

5- If not then leave that process and keep checking the further processes.

### **Sample Inputs and Outputs:**

#### **Example (First fit):**

Sample Input:  $\text{blockSize}[] = \{100, 500, 200, 300, 600\};$

$\text{processSize}[] = \{212, 417, 112, 426\};$  Sample Output:

Process No.	Process Size	Block no.
-------------	--------------	-----------

1	212	2
---	-----	---

2	417	5
---	-----	---

3	112	2
---	-----	---

4	426	Not Allocated
---	-----	---------------

#### **Example (Best fit):**

Sample Input:  $\text{blockSize}[] = \{100, 500, 200, 300, 600\};$

$\text{processSize}[] = \{212, 417, 112, 426\};$  Sample Output:

Process No.	Process Size	Block no.
-------------	--------------	-----------

1	212	4
---	-----	---

2	417	2
---	-----	---

3	112	3
---	-----	---

4	426	5
---	-----	---

#### **Example (Worst fit):**

Sample Input :  $\text{blockSize}[] = \{100, 500, 200, 300, 600\};$

$\text{processSize}[] = \{212, 417, 112, 426\};$  Sample Output:

Process No.	Process Size	Block no.
-------------	--------------	-----------

1	212	5
---	-----	---

2	417	2
---	-----	---

3	112	5
---	-----	---

4	426	Not Allocated
---	-----	---------------

**Viva - Questions:**

1. Difference between Logical and Physical Address Space?
2. Describe first-fit, best-fit strategies and worst fit strategies for disk space allocation, with their merits and demerits.
3. What is the impact of fixed partitioning on fragmentation?
4. In fixed sized partition, the degree of multi programming is bounded by \_\_\_\_\_
5. The first fit, best fit and worst fit are strategies to select a \_\_\_\_\_

## Experiment 7

**Aim:** Write a program to implement reader/writer problem using semaphore.

### Theory: Readers-Writers problem Problem

parameters:

- One set of data is shared among a number of processes
- Once a writer is ready, it performs its write. Only one writer may write at a time
- If a process is writing, no other process can read it
- If at least one reader is reading, no other process can write
- Readers may not write and only read

### Solution when Reader has the Priority over Writer

Here priority means, no reader should wait if the share is currently opened for reading.

Three variables are used: mutex, wrt, readcnt to implement solution

1. semaphore mutex, wrt; // semaphore mutex is used to ensure mutual exclusion when readcnt is updated i.e. when any reader enters or exit from the critical section and semaphore wrt is used by both readers and writers
2. int readcnt; // readcnt tells the number of processes performing read in the critical section, initially 0

### Writer process:

1. Writer requests the entry to critical section.
2. If allowed i.e. wait() gives a true value, it enters and performs the write. If not allowed, it keeps on waiting.
3. It exits the critical section.

### Reader process:

1. Reader requests the entry to critical section.
2. If allowed:
  - it increments the count of number of readers inside the critical section. If this reader is the first reader entering, it locks the **wrt** semaphore to restrict the entry of writers if any reader is inside.
  - It then, signals mutex as any other reader is allowed to enter while others are already reading.
  - After performing reading, it exits the critical section. When exiting, it checks if no more reader is inside, it signals the semaphore “wrt” as now, writer can enter the critical section.
3. If not allowed, it keeps on waiting.

**Pseudocode:****Writer process:** do

```
{  
    // writer requests for critical section  
    wait(wrt);  
  
    // performs the write  
  
    // leaves the critical section  
    signal(wrt);  
} while(true);
```

**Reader process:** do

```
{  
    // Reader wants to enter the critical section  
    wait(mutex);  
  
    // The number of readers has now increased by 1  
    readcnt++;  
  
    // there is atleast one reader in the critical section  
    // this ensure no writer can enter if there is even one reader  
    // thus we give preference to readers here  
    if (readcnt==1)        wait(wrt);  
  
    // other readers can enter while this current reader is inside  
    // the critical section  
    signal(mutex);  
  
    // current reader performs reading here  
    wait(mutex); // a reader wants to leave readcnt--;  
  
    // that is, no reader is left in the critical section,  
    if (readcnt == 0)  
  
        signal(wrt);    // writers can enter  
  
    signal(mutex); // reader leaves
```

```
} while(true);
```

### Sample Inputs and Outputs:

```
File Edit View Terminal Help
student@cc-02-desktop:~$ cd 004
student@cc-02-desktop:~/004$ cc readerw.c -lpthread -o readerw
student@cc-02-desktop:~/004$ ./readerw

value of shared variable is 0
W :written =0
W going to sleep..
R :read=0 Reader no.=1
Reader1:going to sleep...
R :read=0 Reader no.=2
Reader2:going to sleep...
R :read=0 Reader no.=1
Reader1:going to sleep...
R :read=0 Reader no.=2
Reader2:going to sleep...
R :read=0 Reader no.=1
Reader1:going to sleep...
R :read=0 Reader no.=2
Reader2:going to sleep...
R :read=0 Reader no.=1
Reader1:going to sleep...
R :read=0 Reader no.=2
Reader2:going to sleep...
R :read=0 Reader no.=1
Reader1:going to sleep...
R :read=0 Reader no.=2
Reader2:going to sleep...Reader no.=1 livingReader no.=2 living
W :written =2
W going to sleep..
W :written =4
W going to sleep..Writer is livingstudent@cc-02-desktop:~/004$
```

**Viva - Questions:**

1. Define the critical section problem and explain the necessary characteristics of a correct solution.
2. What do understand by Race Condition?
3. Define semaphore and its limitations.
4. Define monitor.
5. What are classical problems of process synchronization?



## Experiment 8

**Aim:** Write a program to implement Banker's algorithm for deadlock avoidance.

**Theory:** Banker's algorithm is a deadlock avoidance algorithm. It is named so because this algorithm is used in banking systems to determine whether a loan can be granted or not.

Consider there are  $n$  account holders in a bank and the sum of the money in all of their accounts is  $S$ . Every time a loan has to be granted by the bank, it subtracts the loan amount from the total money the bank has. Then it checks if that difference is greater than  $S$ . It is done because, only then, the bank would have enough money even if all the  $n$  account holders draw all their money at once.

Banker's algorithm works in a similar way in computers. Whenever a new process is created, it must exactly specify the maximum instances of each resource type that it needs.

Let us assume that there are  $n$  processes and  $m$  resource types. Some data structures are used to implement the banker's algorithm. They are:

1. **Available:** It is an array of length  $m$ . It represents the number of available resources of each type. If  $\text{Available}[j] = k$ , then there are  $k$  instances available, of resource type  $R_j$ .
2. **Max:** It is an  $n \times m$  matrix which represents the maximum number of instances of each resource that a process can request. If  $\text{Max}[i][j] = k$ , then the process  $P_i$  can request atmost  $k$  instances of resource type  $R_j$ .
3. **Allocation:** It is an  $n \times m$  matrix which represents the number of resources of each type currently allocated to each process. If  $\text{Allocation}[i][j] = k$ , then process  $P_i$  is currently allocated  $k$  instances of resource type  $R_j$ .
4. **Need:** It is an  $n \times m$  matrix which indicates the remaining resource needs of each process. If  $\text{Need}[i][j] = k$ , then process  $P_i$  may need  $k$  more instances of resource type  $R_j$  to complete its task.

$$\text{Need}[i][j] = \text{Max}[i][j] - \text{Allocation}[i][j]$$

### Resource Request Algorithm:

This describes the behaviour of the system when a process makes a resource request in the form of a request matrix. The steps are:

1. If number of requested instances of each resource is less than the need (which was declared previously by the process), go to step 2.
2. If number of requested instances of each resource type is less than the available resources of each type, go to step 3. If not, the process has to wait because sufficient resources are not available yet.
3. Now, assume that the resources have been allocated. Accordingly do,

$$\text{Available} = \text{Available} - \text{Request}_i$$

$$\text{Allocation}_i = \text{Allocation}_i + \text{Request}_i$$

$$\text{Need}_i = \text{Need}_i - \text{Request}_i$$

This step is done because the system needs to assume that resources have been allocated. So there will be less resources available after allocation. The number of allocated instances will increase. The need of the resources by the process will reduce. That's what is represented by the above three operations.

After completing the above three steps, check if the system is in safe state by applying the safety algorithm. If it is in safe state, proceed to allocate the requested resources. Else, the process has to wait longer.

### **Safety Algorithm:**

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

$$\text{Work} = \text{Available}$$

$$\text{Finish}[i] = \text{false for } i = 0, 1, \dots, n - 1.$$

This means, initially, no process has finished and the number of available resources is represented by the Available array.

2. Find an index **i** such that both

$$\text{Finish}[i] == \text{false}$$

$$\text{Need}_i \leq \text{Work}$$

If there is no such **i** present, then proceed to step 4.

It means, we need to find an unfinished process whose need can be satisfied by the available resources. If no such process exists, just go to step 4.

3. Perform the following:

Work = Work + Allocation;

Finish[i] = true; Go

to step 2.

When an unfinished process is found, then the resources are allocated and the process is marked finished. And then, the loop is repeated to check the same for all other processes.

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

That means if all processes are finished, then the system is in safe state.

### Sample Inputs and Outputs:

Example 1 for safe sequence:

Input:

Total Resources	R1	R2	R3
	10	5	7

Process	Allocation			Max		
	R1	R2	R3	R1	R2	R3
P1	0	1	0	7	5	3
P2	2	0	0	3	2	2
P3	3	0	2	9	0	2
P4	2	1	1	2	2	2

Output : Safe sequences are:

P2--> P4--> P1--> P3

P2--> P4--> P3--> P1

P4--> P2--> P1--> P3

P4--> P2--> P3--> P1

There are total 4 safe-sequences

Example 2

For non safe state

Enter the number of process:3

Enter the 0th process allocated resources:1 2 3  
Enter the 0th process maximum resources:4 5 6  
Enter the 1th process allocated resources:3 4 5  
Enter the 1th process maximum resources:6 7 8  
Enter the 2th process allocated resources:1 2 3  
Enter the 2th process maximum resources:3 4 5  
Enter the available vector:10 12 11

The state sequence :SYSTEM IS NOT IN A SAFE STATE **Viva - Questions:**

1. What necessary conditions can lead to a deadlock situation in a system?
2. What factors determine whether a detection-algorithm must be utilized in a deadlock avoidance system?
3. What is a Safe State and its' use in deadlock avoidance?
4. What are the Methods for Handling Deadlocks?
5. What is resource allocation graph?
6. What are the deadlock avoidance algorithms?
7. Recovery from Deadlock?
8. When does deadlock happen? How does Banker's algorithm avoid the deadlock condition?

## Experiments beyond the syllabus

**Aim:** 1. Installation of Linux operating system.

Steps to install linux are mentioned in the introduction to lab of manual.

**Aim:** 2. Introduction of Basics Files, directories, system Commands of Linux. **Theory:**

Linux basic commands

**Name:** **cp**

Syntax: cp [options] file1 file2

Description: Copy the file file1 to file2.

Common options:-r copy the entire directory Example:

aaa copy (existing), and named bbb:

```
cp aaa bbb
```

**Name:** **mv**

Syntax: mv [options] source ... directory

Description: Rename the file, or the number of files to another directory.

Example: aaa renamed as bbb:

```
mv aaa bbb
```

**Name:** **rm**

Syntax: rm [options] name ...

Description: delete files and directories.

Commonly used options:-f to force delete files

Example: Remove all but the suffix named c file rm

```
*. c
```

**Name:** **cat**

Syntax: cat [options] [file-list]

Description: standard output connection, display a list of files in the file-list file

Example 1: Displays the contents of file1 and file2 cat file1 file2

Example 2: file1 and file2 merged into file3 cat

```
file1 file2> file3
```

**Name: more**

Syntax: more [options] [file-list]

Description: standard output is connected to the paging file in the file list file-list

Example: paging file AAA more AAA

**Name: head**

Syntax: head [options] [file-list]

Description: Display the initial part of the file in the list of files in the file-list, the default display

10 lines;

Example: the initial part of the file AAA head

AAA

**Name: tail**

Syntax: tail [options] [file-list]

Description: Displays the tail of the list of files in the file-list file; default display 10 lines;

Example: tail file AAA tail AAA

**Name: ln**

Syntax: ln [options] existing-file new-file ln

[options] existing-file-list directory

Description: create a link named "existing-file" new-file, Created with the same name for each file contained in the existing-file-list "link in the directory catalog

Commonly used options:-f, regardless of whether the new-file exists, create links -S to create a soft link

Example 1: To establish the soft connection temp.soft, point Chapter3 ln-s

Chapter3 temp.soft

Example 2: for all the files and subdirectories in the examples directory to create a soft connection

ln-s ~ / linuxbook / examples / \* / home / faculty / linuxbook / examples

**Name: chmod**

Syntax: `chmod [option] mode file-list`

Description: read, write, or execute permissions change or set the parameters in the filelist

Example: Add file job executable permissions `chmod`

`+ x job`

**Name: tar**

Syntax: `tar [option] [files]`

Description: The backup file. Can be used to create a backup file or restore a backup file.

Example 1: a backup test directory the file named `test.tar.gz`, executable commands:

`tar-zcvf test.tar.gz test`

Example 2: Unzip the the associated `test.tar.gz` file, executable commands:

`tar-zxvf test.tar.gz`

**Name: cd**

Syntax: `cd [directory]`

Description: The current working directory to the directory specified by "directory".

Example: enter the directory `/usr/bin/`:

`cd /usr/bin`

**Name: ls**

Syntax: `ls [options] [pathname-list]`

Description: display the file name within the directory and file name specified in the "pathname-list"

Example: List all names in the current working directory is s at the beginning of the file:

`ls s *`

**Name: pwd**

Syntax: `pwd`

Description: Displays the absolute path of the current directory.

**Name: mkdir**

Syntax: `mkdir [options] dirName`

Description: create name is dirName subdirectory.

Example: In the working directory, create a subdirectory named AA:

mkdir AA

**Name: rmdir**

Syntax: rmdir [-p] dirName

Description: delete empty directories.

Example: to delete the working directory, subdirectory named AA:

rmdir AA

**Name: echo**

Syntax: echo \$ variable

Description: Displays the value of the variable variable.

Example 1: Display the current user's PATH value echo

\$ PATH

**Name: ps**

Syntax: \$ ps [options]

Description: The active process is used to view the current system

Example 1: display all current processes ps-aux

**Name: kill**

Syntax: \$ kill [-signal] pid

Description: terminates the specified process

Example 1: the process of termination of 1511

kill 1511

**Name: df**

Syntax: df [option(s)] [directory]

Description: The df (disk free) command, when used without any options, displays information about the total disk space, the disk space currently in use, and the free space on all the mounted drives. If a directory is specified, the information is limited to the drive on which that directory is located.

Commonly used options: -H shows the number of occupied blocks in gigabytes, megabytes, or kilobytes — in human-readable format

Commonly used options: -t Type of file system (ext2, nfs, etc.)

**Name: du**



Syntax: du [option(s)] [path]

Description: This command, when executed without any parameters, shows the total disk space occupied by files and subdirectories in the current directory.

Commonly used options: -a Displays the size of each individual file

Commonly used options: -h Output in human-readable form

Commonly used options: -s Displays only the calculated total size

**Name: free**

Syntax: free [option(s)]

Description: The command free displays information about RAM and swap space usage, showing the total and the used amount in both categories.

Commonly used options: -b Output in bytes

Commonly used options: -k Output in kilobytes

Commonly used options: -m Output in megabytes

**Name: date**

Syntax: date [option(s)]

Description: This simple program displays the current system time. If run as root, it can also be used to change the system time. Details about the program are available in date.

Example 1: Running date date

Thu Feb 8 16:47:32 MST 2018

## Experiments beyond the syllabus

**Aim:** 3 i) Write a script to find the greatest of three numbers (numbers passed as command line parameters) ii) Write a script to check whether the given no. is even/odd iii) Write a script to check whether the given number is prime or not iv) Write a script to check whether the given input is a number or a string  
v) Write a script to compute no. of characters and words in each line of given file

**Theory:** This group of programs is based on flow control statement.

The **if...else...fi** statement is the next form of control statement that allows Shell to execute statements in a controlled way and make the right choice.

```
if [ expression ] then
```

```
    Statement(s) to be executed if expression is true else
```

```
    Statement(s) to be executed if expression is not true fi
```

The Shell expression is evaluated in the above syntax. If the resulting value is true, given statement(s) are executed. If the expression is false, then no statement will be executed.

### Example 1 (if)

```
a=10 b=20
```

```
if [ $a == $b ] then
```

```
    echo "a is equal to b" else
```

```
    echo "a is not equal to b"
```

```
fi
```

### Output

```
a is not equal to b
```

There are many different ways that a conditional statement can be used.

Below is the list of available mnemonics for conditional statements in shell:

Arithmetic operator	Description
==	Equal
>=	Greater Than or Equal
>	Greater Than
<=	Less Than or Equal
<	Less Than
!=	Not Equal

String Comparison	Description
Str1 = Str2	Returns true if the strings are equal
Str1 != Str2	Returns true if the strings are not equal
-n Str1	Returns true if the string is not null
-z Str1	Returns true if the string is null

Numeric Comparison	Description
expr1 -eq expr2	Returns true if the expressions are equal
expr1 -ne expr2	Returns true if the expressions are not equal
expr1 -gt expr2	Returns true if expr1 is greater than expr2
expr1 -ge expr2	Returns true if expr1 is greater than or equal to expr2
expr1 -lt expr2	Returns true if expr1 is less than expr2
expr1 -le expr2	Returns true if expr1 is less than or equal to expr2
! expr1	Negates the result of the expression

File Conditionals	Description
-d file	True if the file is a directory
-e file	True if the file exists (note that this is not particularly portable, thus -f is generally used)
-f file	True if the provided string is a file
-g file	True if the group id is set on a file
-r file	True if the file is readable
-s file	True if the file has a non-zero size
-u	True if the user id is set on a file
-w	True if the file is writable
-x	True if the file is an executable

### Experiments beyond the syllabus

**Aim:** 4 i) Write a script to calculate the average of n numbers.

ii) Write a script to print the Fibonacci series up to n terms. iii)

Write a script to calculate the factorial of a given number. iv) Write

a script to calculate the sum of digits of the given number. v) Write

a script to check whether the given string is a palindrome

**Theory:** This group of programs is based on loop control statement.

A loop is a powerful programming tool that enables you to execute a set of commands repeatedly. Types of loops available to shell programmers -

- The for loop
- The while loop
- The until loop

**For** and **while** loop executes the given commands until the given condition remains true; the **until** loop executes until a given condition becomes true.

## For Loop

The for loop is a looping statement that uses the keyword *for* to declare a repetitive statement. *for* loops iterate through a set of values until the list is exhausted

The bash supports different syntaxes for the *for* loop statement:

### Syntax 1: For-in Structure

```
for <varName> in <list> do
```

```
#### your statement here done
```

This syntax starts with the keyword *for*, then followed by a variable name, the keyword *in* and the list of possible values for the variable. Each value in the list will be separated with a space and the start of the code lines that will be repeated is defined in the *do* and ends with a *done* keyword.

### Example 1 (for loop) for i in

```
1 2 3 4 5 do echo "Looping  
... number $i" done
```

### Output

```
Looping .... number 1  
Looping .... number 2  
Looping .... number 3  
Looping .... number 4  
Looping .... number 5
```

**Example 2 (for loop)** for i in  
hello 1 \* 2 goodbye do echo  
"Looping ... i is set to \$i" done

### Output

Looping ... i is set to hello  
Looping ... i is set to 1  
Looping ... i is set to (name of first file in current directory)  
... etc ...  
Looping ... i is set to (name of last file in current directory)  
Looping ... i is set to 2  
Looping ... i is set to goodbye

## While Loop

The while statement is a type of repetitive structure in bash that utilizes the keyword while. Unlike the C-type syntax of *for* looping structure, the *while* repetitive control structure separates the initialization, Boolean test and the increment/decrement statement.

### Syntax: While Structure

```
<initialization>  
while(condition) do  
    ###your code goes here  
<increment/decrement> done
```

### Example (While loop)

```
result=0 input=0  
var=1  
while((var <= 5))  
do  
printf "Input integer %d : " $var read  
input  
result=$((result+input))  
var=$((var+1)) done  
echo "the result is " $result
```

### Output

```

lime@lime-Lenovo-G480:~/projectfarm/sample2$ ./loopwFile.sh
Input integer 1 : 2
Input integer 2 : 5
Input integer 3 : 6
Input integer 4 : 3
Input integer 5 : 0
the result is 16
lime@lime-Lenovo-G480:~/projectfarm/sample2$ █

```

Looking at our example, the *while* statement starts with an initialization of our counter variable *var*. Next, the Boolean test is declared after the keyword *while* and the set of statements to be repeated will be declared inside the *do* and *done* statements. In while statements, the interpreter will only start and execute repetition of codes if the Boolean test result is true. On the other hand, the looping statement will only terminate the iteration of codes when the Boolean expression results to false.

## Until Loop

Another type of looping statement that the bash supports is the *until* structure.

The *until* statement executes every command inside the loop until the boolean expression declared results to false. It is the complete opposite of the while statement. **Syntax: Until Loop** until ((<conditional\_statement>)) do

####set of commands done

### Example (Until loop)

```

result=0
input=0 var=1
until((var > 5))
do
printf "Input integer %d : " $var read
input
result=$((result+input))
var=$((var+1)) done
echo "the result is " $result

```

### Output

```
lime@lime-Lenovo-G480:~$ cd projectfarm/sample2
lime@lime-Lenovo-G480:~/projectfarm/sample2$ ./until.sh
Input integer 1 : 2
Input integer 2 : 3
Input integer 3 : 4
Input integer 4 : 5
Input integer 5 : 10
the result is 24
lime@lime-Lenovo-G480:~/projectfarm/sample2$ █
```