LAB MANUAL

CIC-353
Operating Systems Lab

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

MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY

VISION

To attain global excellence through education, innovation, research, and work ethics with the commitment to serve humanity.

MISSION

M1. To promote diversification by adopting advancement in science, technology, management, and allied discipline through continuous learning

M2. To foster moral values in students and equip them for developing sustainable solutions to serve both national and global needs in society and industry.

M3. To digitize educational resources and process for enhanced teaching and effective learning.

M4. To cultivate an environment supporting incubation, product development, technology transfer, capacity building and entrepreneurship.

M5. To encourage faculty-student networking with alumni, industry, institutions, and other stakeholders for collective engagement.

Department of Computer Science & engineering

VISION

"To attain global excellence through education, innovation, research, and work ethics in the field of Computer Science and engineering with the commitment to serve humanity."

MISSION

M1 To lead in the advancement of computer science and engineering through internationally recognized research and education.
M2 To prepare students for full and ethical participation in a diverse society and encourage lifelong learning.
M3 To foster development of problem solving and communication skills as an integral component of the profession.
M4 To impart knowledge, skills and cultivate an environment supporting incubation, product development, technology transfer, capacity building and entrepreneurship in the field of computer science and engineering.
M5 To encourage faculty, student's networking with alumni, industry, institutions, and other stakeholders for collective engagement.

Course Objectives

Course Objectives	
1.	To understand the basics of OS and their functions. To learn the scheduling policies of various operating system.
2.	learn memory management methods.
3.	To understand the characterization of deadlock, system deadlock, preventing deadlock, avoiding deadlock and related concepts.
4.	To understand the meaning of a file, structure of the directories, file structure system and implementation, free-space management.

CIC353.1	Understand the role of operating system in computing device, and ability to understand paging and segmentation methods of memory binding and their pros & cons.
CIC353.2	Understand scheduling of process over a processor. Ability to use concepts of semaphore and its usage in process synchronization.
CIC353.3	Ability to synchronize programs and make the system deadlock free.
CIC353.4	Ability to understand the system like access methods, directory structures, file space allocation in disk and free space management in disk. Ability to understand disk scheduling and disk recovery procedure.

	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**
CIC353.1	**3**	**3**	**2**	**-**	**3**	**-**	**-**	**-**	**-**	**-**	**-**	**-**
CIC353.2	**3**	**3**	**-**	**-**	**2**	**-**	**-**	**-**	**-**	**-**	**-**	**-**
CIC353.3	**3**	**2**	**3**	**-**	**2**	**-**	**-**	**-**	**-**	**-**	**-**	**-**
CIC353.4	**3**	**3**	**-**	**-**	**2**	**-**	**-**	**-**	**-**	**-**	**-**	**-**

	PSO1	**PSO2**	**PSO3**
CIC353.1	**1**	**1**	**1**
CIC353.2	**1**	**-**	**1**
CIC353.3	**1**	**1**	**1**
CIC353.4	**1**	**1**	**1**

INDEX

1. INTRODUCTION TO OPERATING SYSTEM LAB

1.1 Introduction to Operating System

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.

1.1.1 Linux

In this lab we will be working in Linux (also known as GNU/Linux), which 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.

1.1.2 Design

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 that 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).

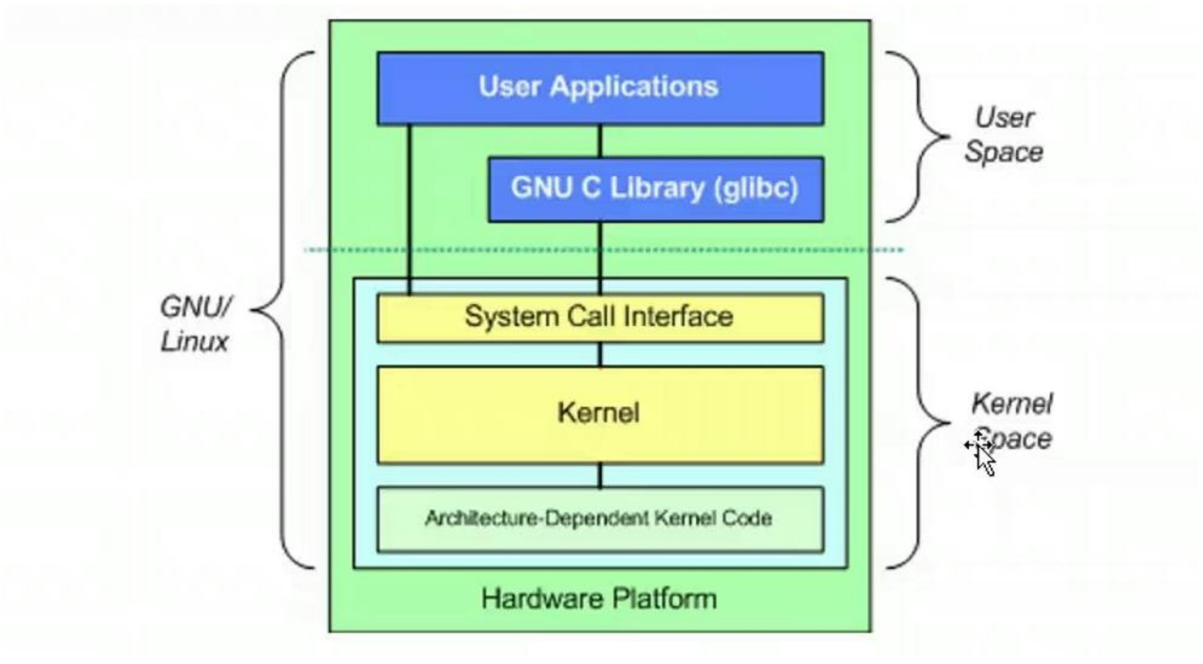


Figure 1:- Linux Architecture

Linux can be controlled by one or more of a text-based command line interface (CLI), GUI, or through controls on the device itself (like on embedded machines). Desktop machines have 3 popular user interfaces (UIs): KDE, GNOME, and Xfce. These UIs run on top of X, which provides network transparency, enabling a graphical application running on one machine to be displayed and controlled from another (that's like running a game on your computer but your friend's computer can control and see the game from his computer). The window manager provides a means to control the placement and appearance of individual application windows, and interacts with the X window system.

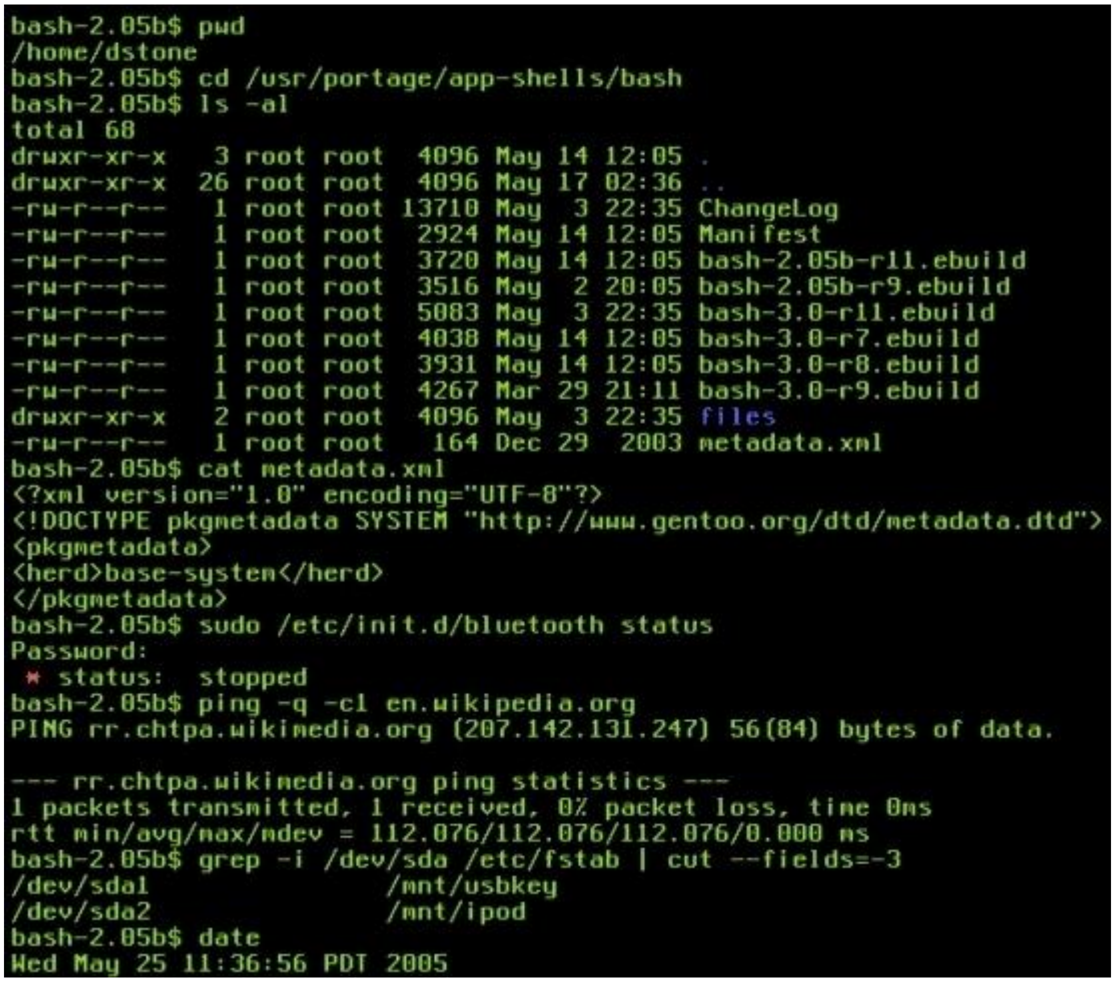
A Linux system usually provides a CLI of some sort through a shell. Linux distros for a server might only use a CLI and nothing else. Most low-level Linux components use the CLI exclusively. The CLI is particularly suited for automation of repetitive or delayed tasks, and provides very simple inter-process communication. A graphical terminal is often used to access the CLI from a Linux desktop.

1.1.2.1 Linux Shell

A shell is a special user program that provides an interface for the user to use operating system services. Shell accepts human-readable commands from users and converts them into something which the kernel can understand. It is a command language interpreter that executes commands read from input devices such as keyboards or from files. The shell gets started when the user logs in or starts the terminal.

Command Line Shell

Shell can be accessed by users using a command line interface. A special program called Terminal in Linux/macOS, or Command Prompt in Windows OS is provided to type in the human-readable commands such as "cat", "ls" etc. and then it is being executed. The result is then displayed on the terminal to the user. A terminal in Ubuntu 16.4 system looks like this –



```
bash-2.05b$ pwd
/home/dstone
bash-2.05b$ cd /usr/portage/app-shells/bash
bash-2.05b$ ls -al
total 68
drwxr-xr-x   3 root root  4096 May 14 12:05 .
drwxr-xr-x  26 root root  4096 May 17 02:36 ..
-rw-r--r--   1 root root 13710 May  3 22:35 ChangeLog
-rw-r--r--   1 root root  2924 May 14 12:05 Manifest
-rw-r--r--   1 root root  3720 May 14 12:05 bash-2.05b-r11.ebuild
-rw-r--r--   1 root root  3516 May  2 20:05 bash-2.05b-r9.ebuild
-rw-r--r--   1 root root  5083 May  3 22:35 bash-3.0-r11.ebuild
-rw-r--r--   1 root root  4038 May 14 12:05 bash-3.0-r7.ebuild
-rw-r--r--   1 root root  3931 May 14 12:05 bash-3.0-r8.ebuild
-rw-r--r--   1 root root  4267 Mar 29 21:11 bash-3.0-r9.ebuild
drwxr-xr-x   2 root root  4096 May  3 22:35 files
-rw-r--r--   1 root root   164 Dec 29  2003 metadata.xml
bash-2.05b$ cat metadata.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pkgmetadata SYSTEM "http://www.gentoo.org/dtd/metadata.dtd">
<pkgmetadata>
<herd>base-system</herd>
</pkgmetadata>
bash-2.05b$ sudo /etc/init.d/bluetooth status
Password:
 * status:  stopped
bash-2.05b$ ping -q -c1 en.wikipedia.org
PING rr.chtpa.wikimedia.org (207.142.131.247) 56(84) bytes of data.

--- rr.chtpa.wikimedia.org ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 112.076/112.076/112.076/0.000 ms
bash-2.05b$ grep -i /dev/sda /etc/fstab | cut --fields=-3
/dev/sda1               /mnt/usbkey
/dev/sda2               /mnt/ipod
bash-2.05b$ date
Wed May 25 11:36:56 PDT 2005
```

Figure 2:- Linux Command Line

There are several shells are available for Linux systems like –

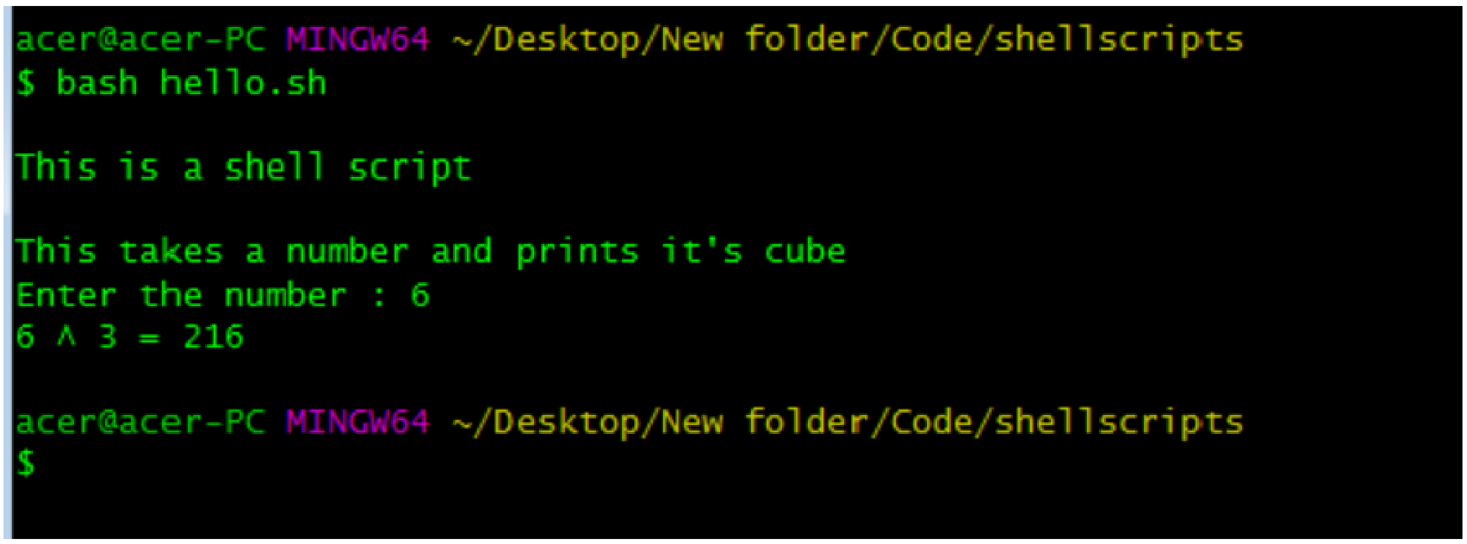
- BASH (Bourne Again Shell) – It is the most widely used shell in Linux systems. It is used as default login shell in Linux systems and in macOS. It can also be installed on Windows OS.
- CSH (C SHell) – The C shell's syntax and its usage are very similar to the C programming language.
- KSH (Korn SHell) – The Korn Shell was also the base for the POSIX Shell standard specifications etc.

- Each shell does the same job but understands different commands and provides different built-in functions

Note: we will be using BASH shell in our lab.

Terminal

A program which is responsible for providing an interface to a user, where user can access the shell. It basically allows users to enter commands and see the output of those commands in a text-based interface. Large scripts that are written to automate and perform complex tasks are executed in the terminal.



```
acer@acer-PC MINGW64 ~/Desktop/New folder/Code/shellscripts
$ bash hello.sh

This is a shell script

This takes a number and prints it's cube
Enter the number : 6
6 ^ 3 = 216

acer@acer-PC MINGW64 ~/Desktop/New folder/Code/shellscripts
$
```

Figure 3: - Bash Shell Terminal

1.1.3 Development

The primary difference between Linux and many other OSs is that the Linux kernel and other components are free and open source software. Free software projects, although developed in a collaborative fashion, are often produced independently of each other. A Linux distribution, commonly called a "distro", is a project that manages a remote collection of Linux-based software, and facilitates installation of a Linux OS. Distros include system software and application software in the form of packages. A distribution is responsible for the default configuration of installed Linux systems, system security, and more generally integration of the different software packages into a coherent whole.

Linux is largely driven by its developer and user communities. Some vendors develop and fund their distros on a volunteer basis. Others maintain a community version of their commercial distros. In many cities and regions, local associations known as Linux Users Groups (LUGs) promote Linux and free software. There are also many online communities that seek to provide support to Linux users and developers. Most distros also have IRC chatrooms or newsgroups for communication. Online forums are another means for support. Linux distros host mailing lists also.

Most Linux distros support dozens of programming languages. The most common collection of utilities for building both Linux applications and OS programs is found within the GNU toolchain, which includes the GNU Compiler Collection (GCC) and the GNU build system. GCC provieds compilers for Ada, C, C++, Java, and Fortran. Most distros also include support for Perl, Ruby, Python and other dynamic languages. The two main frameworks for developing graphical applications are those of GNOME and KDE.

As well as those designed for general purpose use on desktops and servers, distros may be specialized for different purposes including: computer architecture support, embedded systems, stability, security, localization to a specific region or language, targeting of specific user groups, support for real-time applications, or commitment to a given desktop environment. Linux runs on a more diverse range of computer architecture than any other OS.

Although there is a lack of Linux ports for some Mac OS X and Microsoft Windows programs in domains such as desktop publishing and professional audio, applications roughly equivalent to those available for OS X and Windows are available for Linux. Most Linux distros have some sort of program for browsing through a list of free software applications that have already been tested and configured for the specific distro. There are many free software titles popular on Windows that are available for Linux the same way there are a growing amount of proprietary software that is being supported for Linux.

1.2. Vi editor

In this lab we will be using Vi Editor for creating programs. The vi editor is elaborated as visual editor. It is installed in every Unix system. In other words, it is available in all Linux distros. It is user-friendly and works same on different distros and platforms. It is a very powerful application. An improved version of vi editor is vim.

The vi editor has two modes:

Command Mode:
In command mode, actions are taken on the file. The vi editor starts in command mode. Here, the typed words will act as commands in vi editor. To pass a command, you need to be in command mode.

Insert Mode: In insert mode, entered text will be inserted into the file. The Esc key will take you to the command mode from insert mode.
By default, the vi editor starts in command mode. To enter text, you have to be in insert mode, just type 'i' and you'll be in insert mode. Although, after typing i nothing will appear on the screen but you'll be in insert mode. Now you can type anything.

By default vi-editor opens in the command mode. In order to write the code we need to switch to insert mode. Press the '**i**' key when command mode is ON to switch to insert mode.

To exit from insert mode press **Esc** key, you'll be directed to command mode. For saving the code that you have written you need to enter **:wq** after pressing the Esc key.
If you are not sure which mode, you are in, press Esc key twice and you'll be in command mode.

Vi- editor Commands

Exit vi table:

Commands	Action
:wq	Save and quit
:w	Save
:q	Quit
:w fname	Save as fname
ZZ	Save and quit
:q!	Quit discarding changes made
:w!	Save (and write to non-writable file)

To switch from command to insert mode:

Command	Action
I	Start typing before the current character
I	Start typing at the start of current line
A	Start typing after the current character
A	Start typing at the end of current line
O	Start typing on a new line after the current line
O	Start typing on a new line before the current line

To move around a file:

Commands	Action
J	To move down
K	To move up
H	To move left
L	To move right

To jump lines:

Commands	Action
G	Will direct you at the last line of the file
``	Will direct you to your last position in the file

To delete:

Commands	Action
X	Delete the current character
X	Delete the character before the cursor
R	Replace the current character
xp	Switch two characters
dd	Delete the current line
D	Delete the current line from current character to the end of the line
dG	delete from the current line to the end of the file

To repeat and undo:

Commands	Action
U	Undo the last command
.	Repeat the last command

Command to cut, copy and paste:

Commands	Action
Dd	Delete a line
Yy	(yank yank) copy a line
P	Paste after the current line
P	Paste before the current line

Command to cut, copy and paste in blocks:

Commands	Action
<n>dd	Delete the specified n number of lines
<n>yy	Copy the specified n number of lines

Start and end of line:

Commands	Action
Θ	Bring at the start of the current line
^	Bring at the start of the current line
$	Bring at the end of the current line
Dθ	Delete till start of a line
d$	Delete till end of a line

Joining lines:

Commands	Action
J	Join two lines
Yyp	Repeat the current line
Ddp	Swap two lines

Move forward or backward:

Commands	Action
W	Move one word forward
B	Move one word backward
<n>w	Move specified number of words forward
Dw	Delete one word
Yw	Copy one word
<n>dw	Delete specified number of words

Search a string:

Commands	Action
/string	Forward search for given string
?string	Backward search for given string
/^string	Forward search string at beginning of a line
/string$	Forward search string at end of a line
N	Go to next occurrence of searched string
/\<he\>	Search for the word he (and not for there, here, etc.)
/pl[abc]ce	Search for place, plbce, and plcce

Replace all

Syntax:

1. :**<startLine**,endLine> s/**<oldString>**/**<newString>**/g

Example:

Commands	Action
:1,$ s/readable/changed/	Replace forward with backward from first line to the last line
:3,6 s/letters/neww/g	Replace forward with backward from third line to the ninth line

Text buffers:

Commands	Action
"add	Delete current line and put text in buffer a
"ap	Paste the line from buffer a

1.3. Shell Scripting (Using BASH shell)

1.3.1 Writing a shell script

Following steps are required to write shell script:
(1) Use any editor like vi or mcedit to write shell script. Save the file with the extension **'.sh'**

(2) To execute the file we require to take the permission as follows

syntax:

```
chmod permission your-script-name
./ your-script-name
```

Examples: (if the file name is test.sh)

```
$ chmod +x test.sh
$ ./test.sh
```

1.3.2 Variables in Shell

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.

System Variable	**Meaning**
BASH	Our shell name
BASH_VERSION	Our shell Version Name
COLUMNS	No. of columns for our screen
HOME	Our Home Directory
LOGNAME	Our logging Name
OSTYPE	Our OS type
SHELL	Our Shell Name

NOTE: Some of the above settings can be different in your PC/Linux environment. You can print any of the above variables contains as follows:

```
$ echo $USERNAME
$ echo $HOME
```

Example:

1) If you want to print your home directory location then you give command:

a)$ echo $HOME or b)$ echo HOME

(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:

Defining User defined variables (UDV)

To define UDV use following syntax

Syntax:

variable name=value

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

Example:

```
value1 = 40
value2 = 29
```

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. For e.g. valid shell variable are as follows

```
HOME
SYSTEM_VERSION
vech
no
```

(2) Don't put spaces on either side of the equal sign when assigning value to variable. For e.g. In following variable declaration there will be no error

(3) Variables are case-sensitive, just like filename in Linux. For e.g.

```
$ no=10
$ No=11
$ NO=20
$ nO=2
```

Above all are different variable name, so to print value 20 we have to use $ echo $NO and not any of the following

```
$ echo $no # will print 10 but not 20
$ echo $No# will print 11 but not 20
$ echo $nO# will print 2 but not 20
```

(4) You can define NULL variable as follows (NULL variable is variable which has no value at the time of definition) For e.g.

```
$ vech=
$ vech=""
```

Try to print it's value by issuing following command

```
$ echo $vech
```

Nothing will be shown because variable has no value i.e. NULL variable.

(5) Do not use ?,* etc, to name your variable names.

1.3.4 Displaying or accessing the value stored in the variables :

To print or access UDV use following syntax

Syntax:
$variablename

Define variable vech and n as follows:

```
$ value1 = 40
$ n=10
```

To print contains of variable 'vech' type

```
echo $value1
```

echo Command

Use echo command to display text or value of variable.

```
echo [options] [string, variables...]
```

Displays text or variables value on screen.

Options

-n Do not output the trailing new line.
-e Enable interpretation of the following backslash escaped characters in the strings:
\a alert (bell)
\b backspace
\c suppress trailing new line
\n new line
\r carriage return
\t horizontal tab
\\ backslash

For e.g. `$ echo -e "An apple a day keeps away \a\t\tdoctor\n"`

1.3.5 Input command in shell scripting

read command (input command)

In shell scripting, the read command captures user input and stores it in a variable. By default, it reads the entire line of user input until the user presses the Enter key.

Example:

```
$num1
$num2

echo “enter the value of first number”
```

```
read num1
echo “enter the value of second number”
read num2
echo num1
echo num2
```

1.3.6 Shell Arithmetic

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator	`` `expr $a + $b` `` will give 30
- (Subtraction)	Subtracts right hand operand from left hand operand	`` `expr $a - $b` `` will give -10
* (Multiplication)	Multiplies values on either side of the operator	`` `expr $a \* $b` `` will give 200
/ (Division)	Divides left hand operand by right hand operand	`` `expr $b / $a` `` will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder	`` `expr $b % $a` `` will give 0
= (Assignment)	Assigns right operand in left operand	a = $b would assign value of b into a
== (Equality)	Compares two numbers, if both are same then returns true.	[$a == $b] would return false.
!= (Not Equality)	Compares two numbers, if both are different then returns true.	[$a != $b] would return true.

Note :

It is very important to understand that all the conditional expressions should be inside square braces with spaces around them, for example **[$a == $b]** is correct whereas, **[$a==$b]** is incorrect. All the arithmetical calculations are done using long integers.

Example:

```
a=10
b=20

val=`expr $a + $b`
echo "a + b : $val"

val=`expr $a - $b`
```

```
echo "a - b : $val"

val=`expr $a \* $b`
echo "a * b : $val"

val=`expr $b / $a`
echo "b / a : $val"

val=`expr $b % $a`
echo "b % a : $val"
```

output

```
a + b : 30
a - b : -10
a * b : 200
b / a : 2
b % a : 0
```

1.3.7 Relational Operators

Operator	Description	Example
-eq	Checks if the value of two operands are equal or not; if yes, then the condition becomes true.	[$a -eq $b] is not true.
-ne	Checks if the value of two operands are equal or not; if values are not equal, then the condition becomes true.	[$a -ne $b] is true.
-gt	Checks if the value of left operand is greater than the value of right operand; if yes, then the condition becomes true.	[$a -gt $b] is not true.
-lt	Checks if the value of left operand is less than the value of right operand; if yes, then the condition becomes true.	[$a -lt $b] is true.
-ge	Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then the condition becomes true.	[$a -ge $b] is not true.
-le	Checks if the value of left operand is less than or equal to the value of right operand; if yes, then the condition becomes true.	[$a -le $b] is true.

Example:

```
a=10
b=20

if [ $a -eq $b ]
then
  echo "$a -eq $b : a is equal to b"
```

```
else
   echo "$a -eq $b: a is not equal to b"
fi

if [ $a -ne $b ]
then
   echo "$a -ne $b: a is not equal to b"
else
   echo "$a -ne $b : a is equal to b"
fi

if [ $a -gt $b ]
then
   echo "$a -gt $b: a is greater than b"
else
   echo "$a -gt $b: a is not greater than b"
fi

if [ $a -lt $b ]
then
   echo "$a -lt $b: a is less than b"
else
   echo "$a -lt $b: a is not less than b"
fi
if [ $a -ge $b ]
then
   echo "$a -ge $b: a is greater or  equal to b"
else
   echo "$a -ge $b: a is not greater or equal to b"
fi
if [ $a -le $b ]
then
   echo "$a -le $b: a is less or  equal to b"
else
   echo "$a -le $b: a is not less or equal to b"
fi
```

Output

```
10 -eq 20: a is not equal to b
10 -ne 20: a is not equal to b
10 -gt 20: a is not greater than b
10 -lt 20: a is less than b
10 -ge 20: a is not greater or equal to b
10 -le 20: a is less or equal to b
```

1.3.8 Boolean Operators

Operator	Description	Example
!	This is logical negation. This inverts a true condition into false and vice versa.	[! false] is true.
-o	This is logical OR. If one of the operands is true, then the condition becomes true.	[$a -lt 20 -o $b -gt 100] is true.
-a	This is logical AND. If both the operands are true, then the condition becomes true otherwise false.	[$a -lt 20 -a $b -gt 100] is false.

Example:

```
a=10
b=20
if [ $a != $b ]
then
   echo "$a != $b : a is not equal to b"
else
   echo "$a != $b: a is equal to b"
fi
if [ $a -lt 100 -a $b -gt 15 ]
then
   echo "$a -lt 100 -a $b -gt 15 : returns true"
else
   echo "$a -lt 100 -a $b -gt 15 : returns false"
fi
if [ $a -lt 100 -o $b -gt 100 ]
then
   echo "$a -lt 100 -o $b -gt 100 : returns true"
else
   echo "$a -lt 100 -o $b -gt 100 : returns false"
fi
if [ $a -lt 5 -o $b -gt 100 ]
then
   echo "$a -lt 100 -o $b -gt 100 : returns true"
else
   echo "$a -lt 100 -o $b -gt 100 : returns false"
fi
```

Output

```
10 != 20 : a is not equal to b
10 -lt 100 -a 20 -gt 15 : returns true
10 -lt 100 -o 20 -gt 100 : returns true
10 -lt 5 -o 20 -gt 100 : returns false
```

1.3.9 String operators

Operator	Description	Example
=	Checks if the value of two operands are equal or not; if yes, then the condition becomes true.	[$a = $b] is not true.
!=	Checks if the value of two operands are equal or not; if values are not equal then the condition becomes true.	[$a != $b] is true.
-z	Checks if the given string operand size is zero; if it is zero length, then it returns true.	[-z $a] is not true.
-n	Checks if the given string operand size is non-zero; if it is nonzero length, then it returns true.	[-n $a] is not false.
str	Checks if str is not the empty string; if it is empty, then it returns false.	[$a] is not false.

Example:

```
a="abc"
b="efg"

if [ $a = $b ]
then
  echo "$a = $b : a is equal to b"
else
  echo "$a = $b: a is not equal to b"
fi

if [ $a != $b ]
then
  echo "$a != $b : a is not equal to b"
else
  echo "$a != $b: a is equal to b"
fi

if [ -z $a ]
then
  echo "-z $a : string length is zero"
else
  echo "-z $a : string length is not zero"
fi

if [ -n $a ]
then
  echo "-n $a : string length is not zero"
else
  echo "-n $a : string length is zero"
fi
```

```
if [ $a ]
then
   echo "$a : string is not empty"
else
   echo "$a : string is empty"
fi
```

Output

```
abc = efg: a is not equal to b
abc != efg : a is not equal to b
-z abc : string length is not zero
-n abc : string length is not zero
abc : string is not empty
```

2. LAB DETAILS (H/W AND S/W)

Software requirements: Linux’s Shell

Operating System: Fedora

Hardware requirements:

Windows and Linux: Intel 64/32, Athlon 64/32, Opteron processor

2 GB RAM

80 GB hard disk space

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

OPERATING SYSTEMS LAB

Paper Code: CIC - 353	**L T/P C**
Paper: Operating Systems 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 producer-consumer problem using semaphore.
9. Write a program to implement Banker's algorithm for deadlock avoidance.
10. write C program to implement the various File Organization Techniques.

OPERATING SYSTEMS LAB
(Beyond the syllabus prescribed by G.G.S.I.P.U)

Paper Code: CIC-353	**L T/P C**
Paper: Operating Systems Lab	**0 2 1**

List of Experiments:

1. Introduction to Linux and Vi editor.
2. Write a program to find the greatest of three numbers (numbers passed as command line parameters)
3. Write a script to check whether the given no. is even/odd
4. Write a script to calculate the average of n numbers
5. Write a script to check whether the given number is prime or not
6. Write a program to check whether the given input is a number or a string
7. Write a program to compute no. of characters and words in each line of given file
8. Write a program to print the Fibonacci series upto n terms
9. Write a program to calculate the factorial of a given number
10. Write a program to calculate the sum of digits of the given number
11. Write a program to check whether the given string is a palindrome

5. FORMAT OF THE LAB FILE TO BE PREPARED BY THE STUDENTS

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

NAME OF THE LAB

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

Faculty name	Student name Font should be (12", Times Roman)
Font should be (12", Times Roman)	Roll No.:
	Semester:
	Group:

Maharaja Agrasen Institute of Technology, PSP Area,

Sector – 22, Rohini, New Delhi – 110085

Font should be (18", Times Roman)

2. Lab file should include Vision and Mission of the Institute and the Department.

3. Rubrics Evaluation

4. Index for the Lab File is as follows:

OPERATING SYSTEMS LAB

PRACTICAL RECORD

PAPER CODE	**:**	**CIC - 353**
Name of the student	:	
University Roll No.	:	
Branch	:	
Section/ Group	:	

PRACTICAL DETAILS

a) Experiments according to the list provided by GGSIPU

Experiment No.	**Date**	**Experiment Name**	**Marks**					**Total Marks**	**Signature**
			R1	**R2**	**R3**	**R4**	**R5**		
1.									
2.									
3.									
4.									
5.									

b) Experiments beyond the list provided by GGSIPU

Experiment No.	**Date**	**Experiment Name**	**Marks**					**Total Marks**	**Signature**
			R1	**R2**	**R3**	**R4**	**R5**		
1.									
2.									
3.									
4.									
5.									

6. MARKING SCHEME FOR THE PRACTICAL EXAMS

There will be two practical exams in each semester.

i. Internal Practical Exam
ii. External Practical Exam

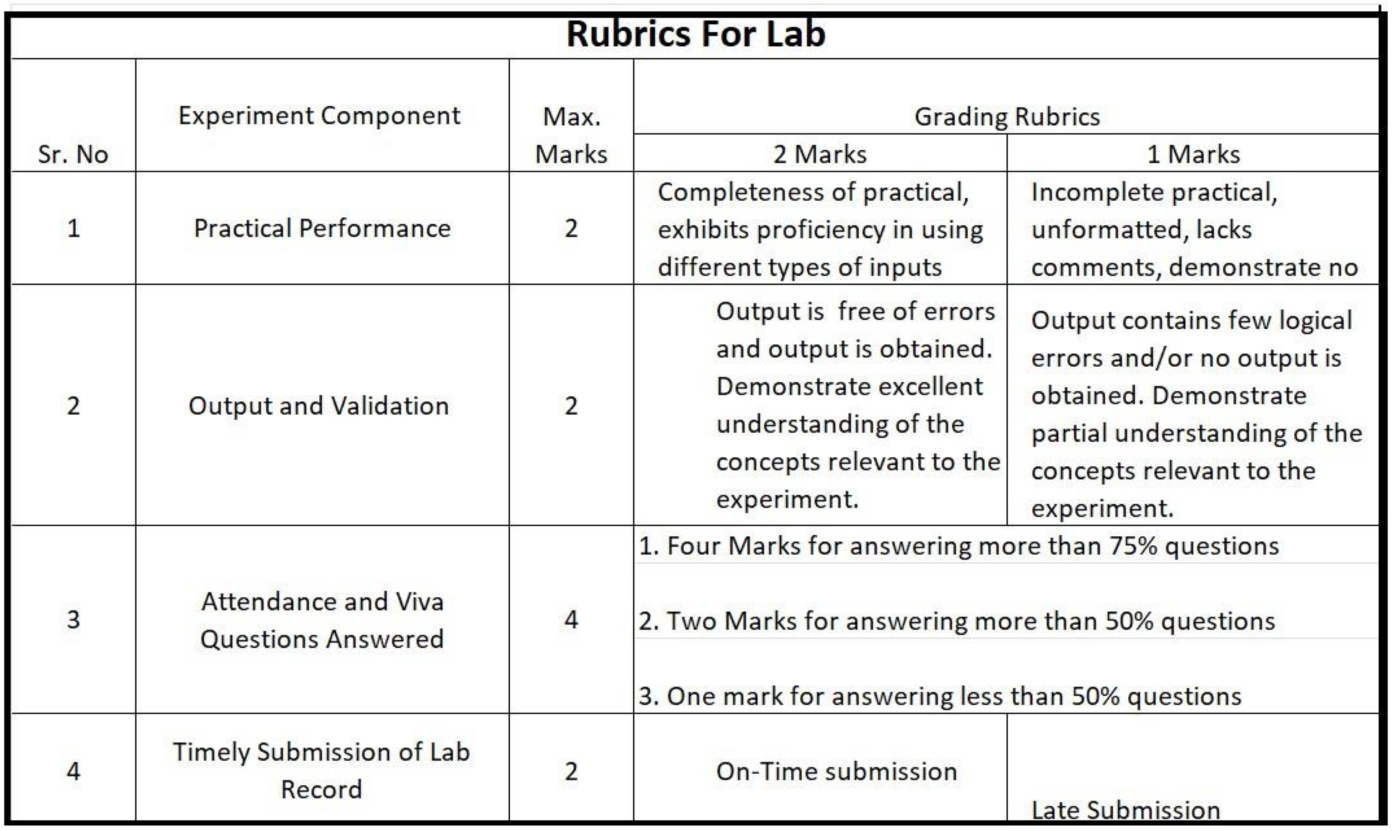
INTERNAL PRACTICAL EXAM

It is taken by the respective faculty of the batch.

MARKING SCHEME FOR THIS EXAM IS:

Total Marks: 40

Division of 10 marks per practical is as follows:



Rubrics For Lab

Sr. No	Experiment Component	Max. Marks	Grading Rubrics	
			2 Marks	1 Marks
1	Practical Performance	2	Completeness of practical, exhibits proficiency in using different types of inputs	Incomplete practical, unformatted, lacks comments, demonstrate no
2	Output and Validation	2	Output is free of errors and output is obtained. Demonstrate excellent understanding of the concepts relevant to the experiment.	Output contains few logical errors and/or no output is obtained. Demonstrate partial understanding of the concepts relevant to the experiment.
3	Attendance and Viva Questions Answered	4	1. Four Marks for answering more than 75% questions 2. Two Marks for answering more than 50% questions 3. One mark for answering less than 50% questions	
4	Timely Submission of Lab Record	2	On-Time submission	Late Submission

Each experiment will be evaluated out of 10 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. Sample Codes

List of GGSIPU

1. **Write a program to implement CPU scheduling for First Come First Serve (FCFS).**

Hint

We are given n processes and their burst times. We need to use the FCFS scheduling method to find the average wait time and average turnaround time. The easiest schedule method is first in, first out (FIFO), which is also written as first come, first served (FCFS). FIFO just puts processes in the queue in the order that they come in. In this, the first process will be run first, and the next process won't begin until the first one is finished.

1. Completion Time: Time at which process completes its execution.
2. Turn Around Time: Time Difference between completion time and arrival time. Turn Around Time = Completion Time – Arrival Time
3. Waiting Time(W.T): Time Difference between turn around time and burst time.
4. Waiting Time = Turn Around Time – Burst Time

Pseudo-Code

When all the processes arrive at 0

```
1- Input the processes along with their burst time (bt) and arrival time(at).
2- As first process that comes need not to wait so
   waiting time for process 1 will be 0 i.e. wt[0] = 0.
3- Find waiting time for all other processes i.e. for
    process i ->
      wt[i] = bt[i-1] + wt[i-1] .
4- Find turnaround time = waiting_time + burst_time
   for all processes.
5- Find average waiting time =
              total_waiting_time / no_of_processes.
6- Similarly, find average turnaround time =
              total_turn_around_time / no_of_processes.
```

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

Shortest job first (SJF), also written as shortest job next, is a scheduling algorithm that chooses the waiting process with the shortest execution time to run next.

Completion Time: Time at which process completes its execution.
Turn Around Time: Time Difference between completion time and arrival time.
Turn Around Time = Completion Time – Arrival Time
Waiting Time(W.T): Time Difference between turnaround time and burst time.
Waiting Time = Turn Around Time – Burst Time

- Sort all the processes according to the arrival time.
- Then select a process that has minimum arrival time and minimum Burst time.
- After completion of the process make a pool of processes that arrives afterward till the completion of the previous process and select that process among the pool which has minimum Burst time.

3. **Write a program to perform priority scheduling.**

Priority scheduling

Hint

Every process is given a first arrival time (less arrival time process first). If two processes have the same arrival time, the goals are compared (highest process first). Also, if two processes have the same importance, compare them based on their process number (first process number less). This process is done over and over while all the others are running.

Algorithm

1. list the processes along with their start time, end time, and importance.
2. The process with the earliest arrival time will be scheduled first. If more than one process has the earliest arrival time, the process with the highest priority will be scheduled first.
3. Now, more processes will be scheduled based on when they arrive and how important they are. That is, we're thinking that lower numbers mean more important things. Put them in order by process number if two of their priorities are the same.

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

Hint

The Round Robin CPU scheduling algorithm is a widely used technique in operating systems for managing the allocation of the central processing unit (CPU) among several processes. This algorithm operates on The algorithm primarily emphasizes the utilization of the Time Sharing approach. The temporal duration during which a process or task is permitted to execute in a preemptive manner is referred to as the time quantum.
Every process or job in the ready queue is allocated the CPU for a specific time period. If the process finishes execution within this time, it will terminate. Otherwise, the process will return to the waiting table and wait for its next turn to complete execution.

Viva Questions (CPU Scheduling)

1. What is the purpose of using FCFS algorithm?
2. Which criteria's are used to evaluate the performance of scheduling algorithms?
3. Define CPU scheduling Algorithms?
4. What is dispatcher?
5. List the various preemptive scheduling algorithms.
6. Explain what a process is.

5. Write a program for page replacement policy using a) LRU b) FIFO c) Optimal.

Hint

Page replacement algorithms are a crucial component of operating systems that manage memory. These algorithms determine which pages in memory should be replaced when a new page needs to be loaded. In operating systems, that employ paging as a memory management technique, the utilization of page replacement algorithms becomes necessary to determine which page should be replaced when a new page is introduced. If a new page is referenced but is not currently residing in memory, a page fault occurs, prompting the Operating System to replace one of the existing pages with the newly required page. Various page replacement algorithms propose distinct methods for determining which page to replace. The primary objective of all algorithms is to minimize the occurrence of page faults.

First In First Out (FIFO)

The First In First Out (FIFO) page replacement technique is a commonly used method in computer operating systems for managing memory. This algorithm involves the operating system maintaining a queue to track all pages currently residing in the memory, with the oldest page positioned at the front of the queue. When a page is required to be replaced, the page located at the front of the queue is chosen for removal.

<u>Algorithm</u>

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.
 i) If set holds less pages than capacity.
   a) Insert page into the set one by one until
      the size  of set reaches capacity or all
      page requests are processed.
   b) Simultaneously maintain the pages in the
      queue to perform FIFO.
   c) Increment page fault
 ii) Else
   If current page is present in set, do nothing.
   Else
     a) Remove the first page from the queue
        as it was the first to be entered in
        the memory
```

```
    b) Replace the first page in the queue with
       the current page in the string.
    c) Store current page in the queue.
    d) Increment page faults.

Return page faults.
```

Least Recently used (LRU)

The Least Recently Used(LRU) algorithm is a type of Greedy algorithm that selects the page to be replaced based on its least recent usage. The concept is founded upon the principle of location of reference, whereby the page that has been least recently used is improbable to be accessed.

<u>Algorithm</u>

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.
 i) If set holds less pages than capacity.
   a) Insert page into the set one by one until
      the size  of set reaches capacity or all
      page requests are processed.
   b) Simultaneously maintain the recent occurred
      index of each page in a map called indexes.
   c) Increment page fault
 ii) Else
   If current page is present in set, do nothing.
   Else
     a) 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.
     b) Replace the found page with current page.
     c) Increment page faults.
     d) Update index of current page.
2. Return page faults.
```

Optimal Page Replacement Algorithm

The Optimal page replacement algorithm is often regarded as the most favorable approach for page replacement in computer systems. The algorithm described herein is designed to replace a page in the frames of secondary memory based on its anticipated future demand relative to other pages. The occurrence of a replacement is triggered by the presence of a page fault. The primary objective of this algorithm is to reduce the occurrence of page faults.

Algorithm

```
1.Create an empty vector to represent the frames.

2.For each page in the page reference sequence:
    a. If the page is found in the current frame, it is considered a hit.
    b. If the page is not found in the current frame, it is considered a miss.
        i. If there is space available in the frames, the page is added to the frame.
        ii. If there is no space available in the frames, find the page that will not be
        used for the longest duration of time in the future.
        iii. Replace the page in the frame with the one that caused the miss.

3.Output the number of hits and misses.
```

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

Hint

First Fit Algorithm

The first fit algorithm allocates a partition in Main Memory by selecting the first available partition that is large enough from the top.

Algorithm

```
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 the current block.
4- If size-of-process <= size-of-block if yes then
   assign and check for the next process.
5- If not then keep checking the further blocks.
```

Best Fit Algorithm

The best fit allocation strategy assigns a process to the smallest available partition that is sufficient to accommodate it, among all the free partitions.

```
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(bockSize[1],
   blockSize[2],.....blockSize[n]) >
   processSize[current], if found then assign
   it to the current process.
```

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

Worst Fit Algorithm

The Worst Fit algorithm assigns a process to the partition that is the largest among the available partitions in the main memory and is sufficient to accommodate the process. If a substantial process occurs at a subsequent step, it may exceed the available memory capacity and be unable to be accommodated.

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

Viva questions (Memory Management)

1. What is demand paging and what are the problems faced to implement demand paging?
2. List various page replacement algorithms used for page replacement.
3. Explain the term virtual memory.
4. List the common strategies used to select a free hole from the set of available holes.
5. Define swapping
6. Define Overlays

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

Hint

A single dataset is distributed among multiple processes. Once an author is prepared, they proceed to engage in the act of writing. The act of writing is limited to a single writer at any one moment. In the context where a process is engaged in writing, it is not possible for any other process to perform a read operation on the same data. If there is a minimum of one individual engaged in the act of reading, it is not possible for any other process to engage in the act of writing. It is possible for readers to engage in passive consumption of text without actively engaging in the act of writing.
Utilization of semaphores is done to ensure that a reader's access to the vital region is uninterrupted, from the moment they enter until they exit. This is crucial in preventing any interference from writers, which could potentially lead to data inconsistency. A semaphore is employed to enforce mutual exclusion between writers and readers using a shared resource, ensuring that authors are prohibited from accessing the resource while one or more readers are currently accessing it. The concept of priority entails that no reader should have a delay if the share is presently accessible for reading.

Solution is implemented using three variables:

mutex, wrt, and readcnt.

The semaphore mutex is employed to provide mutual exclusion during the update of readcnt, which occurs when any reader enters or exits the crucial region. The semaphore wrt is utilized by both readers and writers.
The variable "readcnt" represents the count of processes now engaged in reading within the critical area. Its initial value is set to 0.
The functions pertaining to semaphores are as follows:

1. The function wait() is used to decrement the value of a semaphore.
2. The function signal() is used to increment the value of a semaphore.

Algorithm

Note:- The signal "wrt" is queued on both readers and writers in a way that gives readers priority if writers are also there. This means that no reader is waiting just because a writer asked to go to the critical part.

Writer's Process

1. The writer asks for entry to the critical area.
2. It goes in and writes if it's allowed, which means wait() returns true. It waits if it's not allowed to.
3. It leaves the important part.

```
do {
   // writer requests for critical section
   wait(wrt);

   // performs the write

   // leaves the critical section
   signal(wrt);

} while(true);
```

Reader's Process

1. The reader asks for the entry to the critical part.
2. If it's allowed, it adds one to the number of readers in the critical part. If this reader is the first one to come in, it locks the wrt semaphore so that writers can't come in while readers are already there.

3. Then, it tells the mutex that any other reader can come in while others are still reading.

4. The program leaves the critical part after reading. It checks to see if there are any more readers inside before leaving and sends the "wrt" semaphore, which means that the writer can now enter the critical area.

5. It waits if it's not allowed to.

```
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);
```

8. Write a program to implement producer-consumer problem using semaphore.

Hint

A buffer of a predetermined size is available. A producer has the capability to manufacture a product and afterwards store it within a designated buffer. Consumers have the ability to select and afterwards utilize various things. It is imperative to establish a mechanism wherein the producer is prohibited from placing an item in the buffer simultaneously with the consumer's consumption of any item. The important section in this challenge pertains to the buffer. To solve this problem, pair of counting semaphores is utilized, namely Full and Empty. The variable "Full" is responsible for monitoring the quantity of items currently present in the buffer, whereas the variable "Empty" is responsible for keeping track of the number of slots that are not currently occupied.

Solution to the producer/ consumer problem:

Producer: The number of "empty" goes down by one when the producer makes an item, since one slot is now full. The mutex's value is also lowered so that the user can't get to the buffer. The value of "full" has gone up by 1 now that the producer has put the thing down. The mutex's value also goes up by one because the producer's job is done and the user can now access the buffer.

Consumer: Since the consumer is taking an item out of the buffer, the value of "full" goes down by one and the value of the mutex goes down as well. This means that the creator can't access the buffer right now. The value of "empty" has gone up by 1 since the consumer has used up the thing. The mutex's number has also been raised, which means that the producer can now access the buffer.

Algorithm

mutex = 1
Full = 0 // no lock At first, every slot is empty. So, empty slots are 0 and full slots are n.
Initially, all slots are empty.

Producer Solution

```
do{

//produce an item

wait(empty);
wait(mutex);

//place in buffer

signal(mutex);
signal(full);

}while(true)
```

Consumer Solution

```
do{

wait(full);
wait(mutex);

// consume item from buffer

signal(mutex);
signal(empty);


}while(true)
```

9. Write a program to implement Banker's algorithm for deadlock avoidance.

Hint

The banker's algorithm is a way to distribute resources and avoid deadlocks. It checks to see if the distribution is safe by simulating it for the maximum amount of all resources that can be used. It then does a "s-state" check to see if there are any activities that could happen before deciding if the distribution should be allowed to continue.
To use the Banker's Algorithm, we require data structures. Let 'n' be the number of processes in the system and 'm' be the number of resource types.
Data structure required are:-

Available

- It's a data structure of type 1D array, size of the structure is m, and it stores how many of each type of resource are available.
- If Available[j] = k, it means that there are 'k' instances of the resource type. Rj

Max

- It is a two-dimensional collection of size 'n*m' that shows how much each process in a system can handle at most.
- Max[i, j] = k means that process Pi can only ask for a certain number of cases of resource type Rj.

Allocation

- It is a two-dimensional collection of size 'n*m' that shows how many of each type of resource is currently assigned to each process.
- If allocation[i, j] = k, it means the process Pi has been given 'k' instances of resource type right now Rj.

Need

- It is a two-dimensional collection of size 'n*m' that shows how many resources that each process still needs.
- If [i, j] = k, the process is done. Pi needs 'k' instances of resource type right now. Rj
- Max [i, j] – Allocation [i, j] = Need [i, j]

Note: Allocation lists the resources that are already assigned to process P_i, and $Need_i$ lists the extra resources that process P_i may still need to finish its job.

Algorithm

```
1.Active:= Running U Blocked;

for k=1…r

New_ request[k]:= Requested_ resources[requesting_ process, k];

2. Simulated_ allocation:= Allocated_ resources;

for k=1…..r //Compute projected allocation state
```

```
Simulated_ allocation [requesting _process, k]:= Simulated_ allocation [requesting _process, k]
+ New_ request[k];

3. feasible:= true;

for k=1….r // Check whether projected allocation state is feasible

if Total_ resources[k]< Simulated_ total_ alloc [k] then feasible:= false;

4. if feasible= true

then // Check whether projected allocation state is a safe allocation state

while set Active contains a process P1 such that

For all k, Total _resources[k] – Simulated_ total_ alloc[k]>= Max_ need [l ,k]-Simulated_
allocation[l, k]

Delete Pl from Active;

for k=1…..r

Simulated_ total_ alloc[k]:= Simulated_ total_ alloc[k]- Simulated_ allocation[l, k];

5. If set Active is empty

then // Projected allocation state is a safe allocation state

for k=1….r // Delete the request from pending requests

Requested_ resources[requesting_ process, k]:=0;

for k=1….r // Grant the request

Allocated_ resources[requesting_ process, k]:= Allocated_ resources[requesting_ process, k] +
New_ request[k];

Total_ alloc[k]:= Total_ alloc[k] + New_ request[k];
```

Viva questions(Process Synchronization)

1. **Explain the term deadlock.**
2. **How is mutual exclusion implemented?**
3. **What is deadlock avoidance?**
4. **What is resource allocation graph?**
5. **What are the conditions needed to enter the critical section?**

Shell scripting Programs

Q1. **Script to sum to nos**

```
#
if [ $# -ne 2 ]
then
   echo "Usage - $0   x    y"
   echo "        Where x and y are two nos for which I will print sum”
   exit 1
fi
   echo "Sum of $1 and $2 is `expr $1 + $2`"
```

Q2. **Script to find out biggest number.**

```
   if [ $# -ne 3 ]
   then
            echo "$0: number1 number2 number3 are not given" >&2
      exit 1
   fi
   n1=$1
   n2=$2
   n3=$3
   if [ $n1 -gt $n2 ] && [ $n1 -gt $n3 ]
   then
            echo "$n1 is Bigest number"
   elif [ $n2 -gt $n1 ] && [ $n2 -gt $n3 ]
   then
            echo "$n2 is Bigest number"
   elif [ $n3 -gt $n1 ] && [ $n3 -gt $n2 ]
   then
      echo "$n3 is Bigest number"
   elif [ $1 -eq $2 ] && [ $1 -eq $3 ] && [ $2 -eq $3 ]
   then
            echo "All the three numbers are equal"
   else
      echo "I can not figure out which number is biger"
   fi
```

Q3. **Write script to print nos as 5,4,3,2,1 using while loop.**

```
 i=5
while test $i != 0
do
        echo "$i
"
        i=`expr $i - 1`
done
#
```

Q.4. **Write Script, using case statement to perform basic math operation as follows:**

+ addition
- subtraction
x multiplication
/ division

```
if test $# = 3
then
        case $2 in
         +) let z=$1+$3;;
         -) let z=$1-$3;;
         /) let z=$1/$3;;
         x|X) let z=$1*$3;;
         *) echo Warning - $2 invalied operator, only +,-,x,/ operator allowed
            exit;;
        esac
        echo Answer is $z
else
        echo "Usage - $0   value1  operator value2"
        echo "        Where, value1 and value2 are numeric values"
        echo "               operator can be +,-,/,x (For Multiplication)"
fi
```

Q5. **Write Script to see current date, time, username, and current directory.**

```
echo "Hello, $LOGNAME"
echo "Current date is `date`"
echo "User is `who i am`"
echo "Current direcotry `pwd`"
```

Q6. **Write script to print given number in reverse order, for eg. If no is 123 it must print as 321.**

```
if [ $# -ne 1 ]
then
    echo "Usage: $0   number"
    echo "       I will find reverse of given number"
    echo "       For eg. $0 123, I will print 321"
    exit 1
fi

n=$1
rev=0
sd=0

while [ $n -gt 0 ]
do
    sd=`expr $n % 10`
    rev=`expr $rev \* 10  + $sd`
    n=`expr $n / 10`
done
    echo “Reverse number is $rev"
```

<u>C/C++ Programs</u>

Q1. **Sample code of banker's algorithm**

```
int main()
{
int Max[10][10], need[10][10], alloc[10][10], avail[10], completed[10], safeSequence[10];
int p, r, i, j, process, count;
count = 0;

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

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

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

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

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

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


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

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

        process = -1;

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

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

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

Q2. **Sample code of two-level directory organization**

```
#include<stdio.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir[10];
void 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 \t 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(); }
```

8.Viva – Questions

1. What is an operating system? What are the functions of the operating system?
2. Describe the user's view of the operating system.
3. Name two devices that are used as input devices only. Name two devices that are used as output devices only.
4. Name at least one device that can be used as an input as well as output device.
5. Explain the difference between multi-programming and time sharing.
6. When do we say a system is "multi-programming"? When do we say it is an "on-line" system?
7. Give an example of an operational environment when the system would have to be both multi-programming and on-line system.
8. Explain the difference between platform and environment.
9. What are some of the options available in UNIX command ps? Give description of at least two options.
10. Give examples of at least two applications which in your opinion are real-time applications. Support your example with appropriate rationale.
11. For the file extensions given below indicate the corresponding file type and the usually associated purpose. a. BAT , exe, zip, au b. bin, lib, tex, gif, ar
12. Describe the file system organization. Describe how file hierarchy is managed?
13. Describe at least three file operations.
14. What is the short cut to move up one level from current directory?
15. What is the role of an inode?
16. What is the difference between a program and a process?
17. What is CPU utilization?
18. What is the motivation for Multi-programming and Time sharing.
19. What is "response time"?
20. With the help of a state transition diagram, explain various states of a process.
21. What is a zombie process and how it may manifest itself?
22. Explain the architecture of the simple operating system employing queue data structures?
23. What is the motivation for main memory management?
24. What is the impact of fixed partitioning on fragmentation?
25. Give the relative advantages and disadvantages of load time dynamic linking and run-time dynamic linking. Differentiate them from static linking.
26. Explain the process of linking and loading?
27. Give arguments to support variable partitioning for main memory management.

28. What is meant by virtual memory? With the help of a block diagram explain the data structures used.
29. Describe first-fit and best-fit strategies for disk space allocation, with their merits and demerits.
30. What is a page and what is a frame. How are the two related?
31. What is swapping? Why does one need to swap areas of memory?
32. Discuss virtual memory management scheme. Compare any two page replacement policies
33. Explain the software and hardware methods of implementing page lookup tables.
34. Explain the concept of buffering? How is the double buffering schem.
35. Explain the concepts of spooling.
36. How is the information organized along sectors on a disk?
37. What is a cylinder? Explain different forms of latencies in a disk using diagrams.
38. What is disk scheduling?
39. Explain the difference between shortest seek first and the elevator algorithm scheduling.
40. Which policy would you recommend for scheduling to retrieve inform disc.
41. Give constructional feature of a disk. Explain rotational delay.
42. What is mutual exclusion? Depict a scenario where mutual exclusion is required.
43. What is a dead-lock? List the necessary conditions for a deadlock to occur.
44. Bring out the difference between Deadlock avoidance and deadlock prevention scheme.
45. Explain why having multiple copies of a resource does not prevent deadlocks from happening.
46. Define the critical section problem and explain the necessary characteristics of a correct solution.
47. With the help of the model of resource management, explain the tasks and goals of the resource manager.
48. When does deadlock happen? How does Banker's algorithm avoid the deadlock condition?
49. Explain critical region and mutual exclusion with respect to produce consumer problem.
50. What are semaphores? What are binary semaphores?
51. What is the motivation for establishing inter-process communication?
52. What are system calls? Explain the system call flow with the help of a block diagram.
53. Describe at least three environmental variables that are carried by the child process from the description of parent process.
54. Differentiate between different exec system calls.
55. What are the limitations of a pipe as an IPC mechanism?
56. From point of view of requirements, how is a real-time operating system different from general purpose operating system?
57. What is a hard real-time system and how can we differentiate it from a soft real-time system?
58. How does a device drive in a RTOS differ from the usual?

59. What is a micro-kernel?
60. What is an embedded system? Give at least two examples of embedded systems.
61. Describe the general strategy to define priority structure in a RTOS.
62. How does one determine “schedulability” in RTOS? In which context it is required?
63. List all the files in the current directory with names not exceeding two characters.
64. What is Kernal? Describe the steps involved in booting.
65. Describe how the user and kernel space is divided and used in UNIX operating system.
66. What are system calls? Describe any two system calls and the action take calls are made.
67. How is the system calls used in application programs?
68. Describe the role of a scheduler.
69. Explain user interface and process management in UNIX.
70. What are the Kernel's responsibilities to facilitate I/O transfer?