**PRACTICAL FILE**

**of**

**Operating Systems**



**SUBMITTED BY: SUBMITTED TO:**

***Gaurav Bhardwaj Dr. Prashant Kumar***

***18103034 ASSISTANT PROFESSOR***

***CSE/5th SEMESTER CSE DEPARTMENT***

***GROUP: G2***

**COMPUTER SCIENCE AND ENGINEERING DEPARTMENT**

**DR. B.R. AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY JALANDHAR**

Table of Contents

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | **Topics** | **Page No.** | **Sign.** |
| 1. | Lab 1 | 3 – 5 |  |
| 2. | Lab 2 | 6 – 37 |  |
| 3. | Lab 3 | 38 – 51 |  |
| 4. | Lab 4 | 52 – 59 |  |
| 5. | Lab 5 | 60 – 63 |  |
| 6. | Lab 6 | 64 – 68 |  |
| 7. | Lab 7 | 69 – 74 |  |

# **Lab 1**

1. **Define OS.**

**O**perating **S**ystem (OS) is a program that acts as an interface between a user and a computer hardware. It is a bootstrap program that runs when a system power-ups or reboots which is stored in **ROM** or **EPROM.**

1. **Different types of OS.**

Different types of Operating Systems:-

* **Simple Batch System:**

This is the first and primitive type of OS used for interacting with computer hardwares. There is no as such interaction b/w the user and computer, in which the user enter the input as punch cards/tapes in the computer operators and the computer calculates the job on the given cards as output.

* **Multiprogramming Batch System:**

In this system, CPU takes jobs stored in the memory, if a request of I/O operation is needed then it takes another job. Once a job is done, it then executes the next job. In this system, the CPU never becomes IDLE and keeps on processing.

* **Multiprocessor System:**

As the name says **MULTIPROCESSOR**, the system consists of more than one processor that shares a single physical memory. These all processors work under a single operating system. Advantage of this type of system is that it increases computation and speed.

* **Desktop System:**

Earlier, CPUs and PCs lacked the features needed to protect an operating system from user programs, these are made focusing on higher performance and  peripheral utilization. Now, these are made more user convenience and responsiveness. Development of these types of OSs help in development of OS for mainframe computers.

* **Distributed Operating System:**

The main aim of these systems is to provide powerful and inexpensive microprocessors that can be used for advancement in communication technology. The system consists of multiple computers connected via communication network which helps in providing less price/performance ratios and less load on host computer.

Types:-

Client-Server Systems Peer-to-peer Systems

* **Clustered System:**

These systems are similar to parallel systems in case that these requires two or more connected systems but not that these are handled parallely. This is done for providing high availability and each system can see other systems. The benefit of this is, if in case a connects systems fails due to any reasons, its work can be transferred to other system and users of other systems can see the details of the failing of the system.

**Types:**

**Asymmetric-**

In this a system is made in standby mode to provide and monitor server to others, if server fail, the host becomes the server.

**Symmetric-**

In this two or more hosts are made which are monitoring each other.

**Parallel-**

Parallel clusters allow multiple hosts to access the same data on the shared storage.

* **Realtime Operating System:**

These systems are made to provide maximum time for Critical computation.

**Types:**

**Hard-Real Time Operating Systems-**

That provides maximum time for critical tasks and ensures that these are performed on time.

**Soft-Real Time Operating Systems-**

Similar to Hard -Type, Critical tasks are given priority and are performed ,but no assurity is given that tasks will be performed in given time.

* **Handheld System:**

The system including **Personal Direct Assistants (PDAs)** like cellular phones, digital cameras with connectivity to network as the internet. These systems use less memory(512 **KB** - 8 **MB**),less power,have small display. The developers ensures that system use memory efficiently since less memory is given, and to handle less memory less power consumption is there since they have slow processors.

Some devices also use wireless networks like **BlueTooth** for connecting to other devices and to provide internet. These are used in digital cameras ,media players, calling headsets,etc.

1. **Examples of OS.**

* **Microsoft Window / MacOS / LINUX:**  These OSs are designed and used for Personal Computer / Laptops.
* **Android OS / iOS :** These OSs are used in Mobile Phones and Tablets.
* **Wear OS:** Used in wearable devices like Smart-watches and Smart-bands.

# **Lab 2**

## Part A:

**The introduction to Operating System:**

An Operating System (OS) is an interface between a computer user and computer hardware. An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers. Some popular Operating Systems include **Linux Operating System,** **Windows Operating System,** **Mac OS**, **Android**, etc.

Following are some of important functions of an operating System.

* Memory Management
* Processor Management
* Device Management
* File Management
* Security
* Control over system performance
* Job accounting
* Error detecting aids
* Coordination between other software and users

**The introduction to linux/Unix:**

The Unix operating system is a set of programs that act as a link between the computer and the user.

The computer programs that allocate the system resources and coordinate all the details of the computer's internals is called the **operating system** or the **kernel**.

Users communicate with the kernel through a program known as the **shell**. The shell is a command line interpreter; it translates commands entered by the user and converts them into a language that is understood by the kernel.

There are various Unix variants available in the market. Solaris Unix, AIX, HP Unix and BSD are a few examples. Linux is also a flavor of Unix which is freely available.

**LINUX** is an operating system or a kernel distributed under an open-source license. Its functionality list is quite like UNIX. The kernel is a program at the heart of the Linux operating system that takes care of fundamental stuff, like letting hardware communicate with software.

Linux is an operating system or a kernel which germinated as an idea in the mind of young and bright **Linus Torvalds** when he was a computer science student. He used to work on the **UNIX OS (proprietary software)**and thought that it needed improvements.

This definitely curbed the Linux's popularity as other commercially oriented Operating System Windows got famous. Nonetheless, the open-source aspect of the Linux operating system made it more robust.

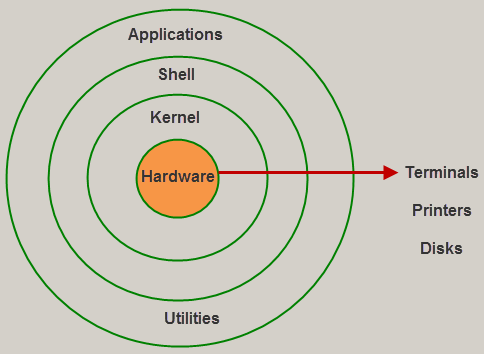
The **main advantage of Linux** was that programmers were able to use the Linux Kernel to design their own custom operating systems. With time, a new range of user-friendly OS's stormed the computer world. Linux now enjoys popularity at its prime, and it's famous among programmers as well as regular computer users around the world. Its main benefits are -

It offers a **free operating system**. You do not have to shell hundreds of dollars to get the OS like Windows!

* Being open-source, anyone with programming knowledge can modify it.
* The Linux operating systems now offer **millions of programs/applications to choose from**, most of them free!
* Once you have Linux installed you no longer need an antivirus! Linux is a highly secure system. More so, there is a global development community constantly looking at ways to enhance its security. With each upgrade, the OS becomes more secure and robust
* Linux is the OS of choice for Server environments due to its stability and reliability (Mega-companies like Amazon, Facebook, and Google use Linux for their Servers). A Linux based server could run non-stop without a reboot for years on end.

**Linux System Architecture**

The Linux Operating System’s architecture primarily has these components: the Kernel, Hardware layer, System library, Shell and System utility.



Architecture of Linux

1. The **kernel** is the core part of the operating system, which is responsible for all the major activities of the LINUX operating system. This operating system consists of different modules and interacts directly with the underlying hardware. The kernel offers the required abstraction to hide application programs or low-level hardware details to the system. The types of Kernels are as follows:

* Monolithic Kernel
* Micro kernels
* Exo kernels
* Hybrid kernels

2. System libraries are special functions, that are used to implement the functionality of the operating system and do not require code access rights of kernel modules.

3. System Utility programs are liable to do individual, and specialized-level tasks.

4. Hardware layer of the LINUX operating system consists of peripheral devices such as RAM, HDD, CPU.

5. The shell is an interface between the user and the kernel, and it affords services of the kernel. It takes commands from the user and executes kernel’s functions. The Shell is present in different types of operating systems, which are classified into two types: command line shells and graphical shells.

The command line shells provide a command line interface, while the graphical line shells provide a graphical user interface. Though both shells perform operations, but the graphical user interface shells perform slower than the command line interface shells.

**System calls:**

The interface between a process and an operating system is provided by system calls. In general, system calls are available as assembly language instructions.

The processes execute normally in the user mode until a system call interrupts it. Then the system call is executed on a priority basis in the kernel mode. After the execution of the system call, the control returns to the user mode and execution of user processes can be resumed.

**Types:**

### **Process Control:** These system calls deal with processes such as process creation, process termination etc.

### **File Management:** These system calls are responsible for file manipulation such as creating a file, reading a file, writing into a file etc.

### **Device Management:** These system calls are responsible for device manipulation such as reading from device buffers, writing into device buffers etc.

### **Information Maintenance:** These system calls handle information and its transfer between the operating system and the user program.

### **Communication**: These system calls are useful for interprocess communication. They also deal with creating and deleting a communication connection.

## Part B:

1. **The concept of SHELL:**

The **shell** is the command interpreter in an operating system such as **Unix** or **GNU/Linux**, it is a program that executes other programs. It provides a computer user an interface to the Unix/GNU Linux system so that the user can run different commands or utilities/tools with some input data.

When the shell has finished executing a program, it sends an output to the user on the screen, which is the standard output device. For this reason, it is referred to as the “**command interpreter**”.

1. **Shell Prompt:**

The prompt, **$**, which is called the **command prompt**, is issued by the shell. While the prompt is displayed, you can type a command.

Shell reads your input after you press **Enter**. It determines the command you want executed by looking at the first word of your input. A word is an unbroken set of characters. Spaces and tabs separate words.

There are two major types of shells −

* **Bourne shell** − If you are using a Bourne-type shell, the **$** character is the default prompt.
* **C shell** − If you are using a C-type shell, the % character is the default prompt.

The Bourne Shell has the following subcategories −

* Bourne shell (sh)
* Korn shell (ksh)
* Bourne Again shell (bash)
* POSIX shell (sh)

The different C-type shells follow −

* C shell (csh)
* TENEX/TOPS C shell (tcsh)

1. **ls command:**

The **ls** command is a command-line utility for listing the contents of a directory or directories given to it via standard input. It writes results to standard output. The **ls** command supports showing a variety of information about files, sorting on a range of options and recursive listing:

* To show the contents of a directory pass the directory name to the **ls** command.
* To show hidden files and folders pass the **-a** option to **ls**
* To append an indicator of the file type to a directory listing pass the **-F** option.
* To show a long listing pass the **-l** option to the **ls** command.
* To sort a directory listing by name pass the **-S** option.
* To sort by modification time pass the **-t** option.
* To show file size in human readable format pass the **-h** option.
* To display one file or folder per line pass the **-1** option.
* To display a recursive listing pass the **-R** option. This causes folders and files within a folder to be listed.

1. **> Command:**

**‘>’** isused to create immediate files and editing.

****

To save editing and exit the file press **<ctrl> + D**, **cat** is generally used to look inside the file but **>** operator it enables you edit it until you press **<ctrl + d>.**

1. **Cat Command:**

The cat (short for “**concatenate**“) command is one of the most frequently used command in Linux/Unix like operating systems. **cat** command allows us to create single or multiple files, view contain of file, concatenate files and redirect output in terminal or files.

Used as: cat [options] [file] …

Eg:



Option:

* Look inside a file as shown above
* Write into file using **>** operator



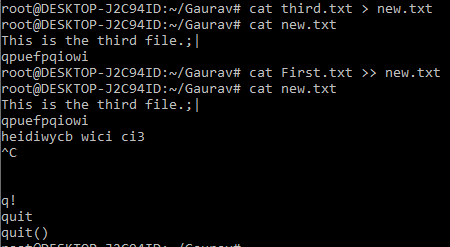
* To see the line no. with file contents we can use **–n** option



* To see contents of file containing very large no. of line using |**more/ |less** option
* To see end of line represented by **$,** using **–e** option



* We can see **tab spaces** using **–T** option, space is shown by **^I**
* We can overwrite/append the contents of one file to other by using **>**/**>>** respectively.



* And many more

1. **Feeding output of one command to another by pipeline:**

A pipe is a form of redirection (transfer of standard output to some other destination) that is used in Linux and other Unix-like operating systems to send the output of one command/program/process to another command/program/process for further processing. You can make it do so by using the pipe character **|**.

Pipe is used to combine two or more commands, and in this, the output of one command acts as input to another command, and this command’s output may act as input to the next command and so on.

Syntax : command\_1 | command\_2 | command\_3 | .... | command\_N

1. **Ways of signing off from LINUX:**

A root user can logout and kill any user session forcefully using the following commands:

a) pkill command – Kill processes by name.

b) kill command – terminate or signal a process.

c) logout command – Logout of a login shell. This command can be used by normal users to end their own session.

**8. Locating Commmands:**

Anything can be searched in system using –I filename. To search for a file that contains two or more words, use an asterisk **(\*)**. For example, **locate -i school\*note** command will search for any file that contains the word “school” and “note”, whether it is uppercase or lowercase. Or we can use **$** operator.

Syntax :

$ find [where to start searching from]

[expression determines what to find] [-options] [what to find]

Options :

● -exec CMD: The file being searched which meets the above criteria and returns 0 for as its exit status for successful command execution.

● -ok CMD : It works same as -exec except the user is prompted first.

● -inum N : Search for files with inode number ‘N’.

● -links N : Search for files with ‘N’ links.

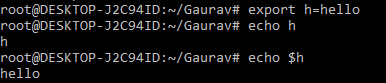
● -name demo : Search for files that are specified by ‘demo’.

● -empty : Search for empty files and directories.

● \(expr \) : True if ‘expr’ is true; used for grouping criteria combined with OR or AND.

● ! expr : True if ‘expr’ is false.

9. **Path and Shell Commands:**







Whenever you run a program from the command line, Bash actually searches through a list of directories looking for the program that you requested. Eg: when you run the **ls** command, Bash doesn't know where this command is located without looking through a list of possible places.

Eg: root@DESKTOP-J2C94ID:~/Gaurav# $PATH=$PATH:~newdir

-bash: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/mnt/c/Program: No such file or directory

10. **Combining Command/operators:**

🡪 The Semicolon (;) Operator

The semicolon (;) operator allows you to execute multiple commands in succession, regardless of whether each previous command succeeds.



🡪 The Logical AND Operator (&&)

If you want the second command to only run if the first command is successful, separate the commands with the logical AND operator, which is two ampersands ( && ). My recommendation to you is using the logical AND operator rather than the semicolon operator most of the time (;). This ensures that you don’t do anything disastrous. Eg:



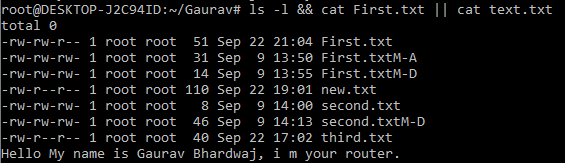
🡪 The Logical OR Operator (||)

Sometimes you might want to execute a second command only if the first command does not succeed. To do this, we use the logical OR operator, or two vertical bars ( || ). Eg:



🡪 Combining Multiple Operators

We can also use combine multiple operators described above on the command line, too. Eg:

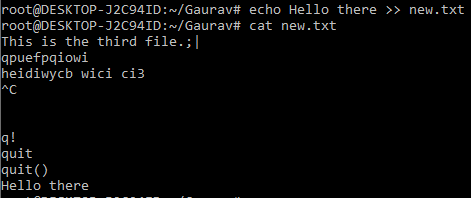


* 1. Echo command:

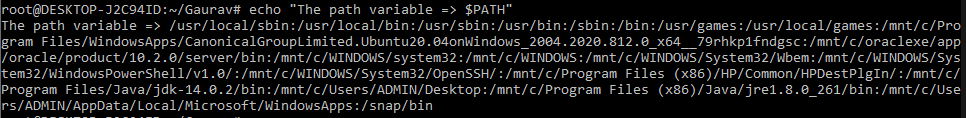
The echo command does one simple job: it prints to the output the argument passed to it.This example:



We can append the output to a file:



We can interpolate environment variables: Beware that special characters need to be escaped with a backslash \. $ for example:



This is just the start. We can do some nice things when it comes to interacting with the shell features.

We can echo the files in the current folder:

We can echo the files in the current folder that start with the letter o:



Any valid Bash (or any shell you are using) command and feature can be used here. You can print your home folder path:

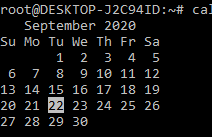


* 1. **Cal and Date command with its different options.**

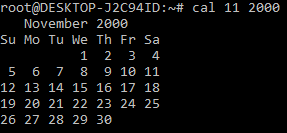
Syntax: **cal [ [ month ] year]**

Rectangular bracket means it is optional, so if used without option, it will display a calendar of the current month and year.

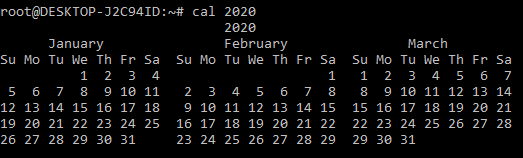
● cal : Shows current month calendar on the terminal.



● cal 08 2000 : Shows calendar of selected month and year.



● cal 2018 : Shows the whole calendar of the year.



● cal 2018 | more : But year may not be visible in the same screen, use more with cal use a spacebar to scroll down.

● cal -3 : Shows calendar of previous, current and next month

DATE: Syntax: **date [OPTION]... [+FORMAT]**

**date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]**

Commands:





* + - **man and help command:**

man command in Linux is used to display the user manual of any command that we can run on the terminal. It provides a detailed view of the command which includes NAME, SYNOPSIS, DESCRIPTION, OPTIONS, EXIT STATUS, RETURN VALUES, ERRORS, FILES, VERSIONS, EXAMPLES, AUTHORS and SEE ALSO.

Every manual is divided into the following sections:

● Executable programs or shell commands

● System calls (functions provided by the kernel)

● Library calls (functions within program libraries

● Games ● Special files (usually found in /dev)

● File formats and conventions eg /etc/passwd

● Miscellaneous (including macro packages and conventions), e.g. groff(7)

● System administration commands (usually only for root)

● Kernel routines [Non standard]

Syntax : $man [OPTION]... [COMMAND NAME]...

If you are new to LINUX operating system and having trouble dealing with the command-line utilities provided by LINUX then you really need to know first of all about the help command which as its name says help you to learn about any built-in command.

help command as told before just displays information about shell built-in commands. Here’s the syntax for it: $help [-dms] [pattern ...]

* 1. **Using escape sequence.**

Certain characters are significant to the shell; we have seen, for example, that the use of double quotes (") characters affect how spaces and TAB characters are treated, for example:

$ echo "Hello World"

Hello World

we can display by : Hello "World"

$ echo "Hello \"World\""

Backslash (\) is special because it is itself used to mark other characters off; we need the following options for a complete shell:

$ echo "This is \\ a backslash"

This is \ a backslash

$ echo "This is \" a quote and this is \\ a backslash"

This is " a quote and this is \ a backslash

* 1. **Printf in Linux**

**printf** command in Linux is used to display the given string, number or any other format specifier on the terminal window. It works the same way as “printf” works in programming languages like C.Syntax: **$printf [-v var] format [arguments]**

Note: printf can have format specifiers, escape sequences or ordinary characters. Format Specifiers: The most commonly used printf specifiers are %s, %b, %d, %x and %f.

## Part C:

**1. Internal and External commands :**

The UNIX system is command-based i.e things happen because of the commands that you key in. All UNIX commands are seldom more than four characters long.  
They are grouped into two categories:

**Internal Commands**: Commands which are built into the shell. For all the shell built-in commands, execution of the same is fast in the sense that the shell doesn’t have to search the given path for them in the PATH variable, and also no process needs to be spawned for executing it.  
Examples: source, cd, fg, etc.

**External Commands**: Commands which aren’t built into the shell. When an external command has to be executed, the shell looks for its path given in the PATH variable, and also a new process has to be spawned and the command gets executed. They are usually located in /bin or /usr/bin. For example, when you execute the “cat” command, which usually is at /usr/bin, the executable /usr/bin/cat gets executed.  
Examples: ls, cat etc.

**2. Commands - passwd, who, uname, tty, sty :**

**passwd:** It is used to create/update passwords for user accounts, it can also change the account or associated password validity period.

**Who:** This command shows information about users who are currently logged in like this.

**uname:** It displays system information such as operating system, network node hostname kernel name, version and release etc.

**tty:** This command of terminal basically prints the file name of the terminal connected to standard input. tty is short of teletype.

**sty:** stty command in Linux is used to change and print terminal line settings. Basically, this command shows or changes terminal characteristics.

**3. Types of files and file system in Linux :**

A file is a smallest unit in which the information is stored. Unix file system has several important features. Types of files are :

**Ordinary files** – An ordinary file is a file on the system that contains data, text, or program instructions, and are used to store your information, such as some text you have written or an image you have drawn. This is the type of file that you usually work with.

**Directories** – Directories store both special and ordinary files. For users familiar with Windows or Mac OS, UNIX directories are equivalent to folders. A directory file contains an entry for every file and subdirectory that it houses. If you have 10 files in a directory, there will be 10 entries in the directory. Each entry has two components.

(1) The Filename

(2) A unique identification number for the file or directory (called the inode number)

**Special Files** – Used to represent a real physical device such as a printer, tape drive or terminal, used for Input/Output (I/O) operations. Device or special files are used for device Input/Output (I/O) on UNIX and Linux systems. They appear in a file system just like an

**Pipes** – UNIX allows you to link commands together using a pipe. The pipe acts a temporary file which only exists to hold data from one command until it is read by another. A Unix pipe provides a one-way flow of data. The output or result of the first command sequence is used as the input to the second command sequence.

**Sockets** – A Unix socket (or Inter-process communication socket) is a special file which allows for advanced inter-process communication. A Unix Socket is used in a client-server application framework

**Symbolic Link** – Symbolic link is used for referencing some other file of the file system. Symbolic link is also known as Soft link. It contains a text form of the path to the file it references. To an end user, symbolic link will appear to have its own name, but when you try reading or writing data to this file, it will instead reference these operations to the file it points to.

Types of file systems are :

**ext2, ext3, ext4**: These are the progressive version of Extended Filesystem (ext), which primarily was developed for MINIX.

**JFS**: The Journaled File System (JFS) was developed by IBM for AIX UNIX which was used as an alternative to system ext.

**ReiserFS**: It was introduced as an alternative to ext3 with improved performance and advanced features. There was a time when SuSE Linux‘s default file format was ReiserFS but later Reiser went out of business and SuSe had no option other than to return back to ext3.

**XFS**: XFS was a high speed JFS which aimed at parallel I/O processing. NASA still usages this file system on their 300+ terabyte storage server.

**Btrfs**: B-Tree File System (Btrfs) focus on fault tolerance, fun administration, repair System, large storage configuration and is still under development. Btrfs is not recommended for Production System.

**4. Directory structure in Linux :**

**/ – Root**

Every single file and directory starts from the root directory. Only root user has write privilege under this directory.

**/bin** – User Binaries

Contains binary executables. Common linux commands you need to use in single-user modes are located under this directory. Commands used by all the users of the system are located here.

**/sbin** – System Binaries

Just like /bin, /sbin also contains binary executables. But, the linux commands located under this directory are used typically by system aministrator, for system maintenance purpose.

**/etc** – Configuration Files

Contains configuration files required by all programs. This also contains startup and shutdown shell scripts used to start/stop individual programs.

**/dev** – Device Files

Contains device files. These include terminal devices, usb, or any device attached to the system.

**/proc** – Process Information

Contains information about system process. This is a pseudo filesystem contains information about running process. For example: /proc/{pid} directory contains information about the process with that particular pid.

**/var** – Variable Files

var stands for variable files. Content of the files that are expected to grow can be found under this directory.

**/tmp** – Temporary Files

Directory that contains temporary files created by system and users. Files under this directory are deleted when system is rebooted.

**/usr** – User Programs

Contains binaries, libraries, documentation, and source-code for second level programs.

/usr/bin contains binary files for user programs. If you can’t find a user binary under /bin, look under /usr/bin. For example: at, awk, cc, less, scp

**/home** – Home Directories

Home directories for all users to store their personal files. For example: /root/Gaurav

**/boot** – Boot Loader Files

Contains boot loader related files. Kernel initrd, vmlinux, grub files are located under /boot

**/lib** – System Libraries

Contains library files that supports the binaries located under /bin and /sbin

Library filenames are either ld\* or lib\*.so.\*

**/opt** – Optional add-on Applications

opt stands for optional. Contains add-on applications from individual vendors.

add-on applications should be installed under either /opt/ or /opt/ sub-directory.

**/mnt** – Mount Directory

Temporary mount directory where sysadmins can mount filesystems.

**/media** – Removable Media Devices

Temporary mount directory for removable devices.

**/srv** – Service Data

srv stands for service. Contains server specific services related data.For example, /srv/cvs contains CVS related data.

**5. HOME variable:**

Environment variables or ENVs basically define behavior of the environment. They can affect the processes ongoing or the programs that are executed in the environment. HOME is an environment variable which gives path of home directory.

**6. Commands - pwd, cd, mkdir, rmdir:**

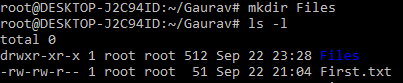
**pwd:** When you first open the terminal, you are in the home directory of your user. To know which directory you are in, you can use the “pwd” command. It gives us the absolute path, which means the path that starts from the root.



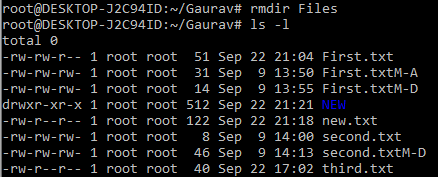
**cd:** cd command in linux known as change directory command. It is used to change current working directory.



**mkdir:** This command is used when you need to create a folder or a directory. For example, if you want to make a directory called “Files”, then you can type “**mkdir** Files”.



**rmdir:** It is used when you need to remove a folder or a directory. For example, if you want to delete a directory called “WINUX”, then you can type “**rmdir** WINUX”.



**7. Absolute and relative pathname :**

A path is a unique location to a file or a folder in a file system of an OS.A path to a file is a combination of / and alpha-numeric characters. An absolute path is defined as specifying the location of a file or directory from the root directory(/). In other words,we can say that an absolute path is a complete path from start of actual file system from / directory.



Relative path is defined as the path related to the present working directly(pwd). It starts at your current directory and never starts with a / .

To be more specific let’s take a look on the below figure in which if we are looking for photos then absolute path for it will be provided as /home/jono/photos but assuming that we are already present in jono directory then the relative path for the same can be written as simple photos.

**8. Using . and .. :**

UNIX offers a shortcut in the relative pathname– that uses either the current or parent directory as reference and specifies the path relative to it. A relative path-name uses one of these cryptic symbols:

**.**(a single dot) - this represents the current directory.



**..**(two dots) - this represents the parent directory.



**9. Commands - Cat, cp, rm, mv, wc, comm, cmp, diff :**

**cat:** It is used to concatenate files and print to stdout.

**cp:** It copies the contents from file1 to file2 and contents of file1 is retained.

**rm:** It Remove files and directories.

**mv:** mv stands for move. It is used to move one or more files or directories from one place to another in file system.

**wc:** The wc (word count) command in Unix/Linux operating systems is used to find out number of newline count, word count, byte and characters count in a files specified by the file arguments.

**comm:** It compares two sorted files line by line and write to standard output; the lines that are common and the lines that are unique.

**cmp:** cmp command in Linux is used to compare the two files byte by byte and helps you to find out whether the two files are identical or not.

**diff:** diff stands for difference. This command is used to display the differences in the files by comparing the files line by line.

**10. Compressing and archiving files (zip, tar) :**

It is useful to store a group of files in one file for easy backup, for transfer to another directory, or for transfer to another computer. It is also useful to compress large files; compressed files take up less disk space and download faster via the Internet.

It is important to understand the distinction between an archive file and a compressed file. An archive file is a collection of files and directories stored in one file.

Archive a Directory :

1. Make a directory on your system and create a text file:

2. mkdir testdir && touch testdir/example.txt

3. Use tar to archive the directory:

tar -cvf testdir.tar testdir/

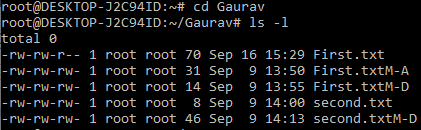
zip is a compression and file packaging utility for Unix/Linux. Each file is stored in single .zip {.zip-filename} file with the extension .zip.

Syntax:  **zip {.zip-filename} {filename-to-compress}**

## Part D:

**1. Basic file attributes: ls –l, -d option.**

-l gives long list of files and directories in current directory.



-ld shows hidden files if present.



**2. File permissions and changing the access rights.**

Every file and directory on your Unix/Linux system is assigned 3 types of owner, given below.

**User**

A user is the owner of the file. By default, the person who created a file becomes its owner. Hence, a user is also sometimes called an owner.

**Group**

A user- group can contain multiple users. All users belonging to a group will have the same access permissions to the file. Suppose you have a project where a number of people require access to a file. Instead of manually assigning permissions to each user, you could add all users to a group, and assign group permission to file such that only this group members and no one else can read or modify the files.

**Other**

Any other user who has access to a file. This person has neither created the file, nor he belongs to a user-group who could own the file. Practically, it means everybody else. Hence, when you set the permission for others, it is also referred as set permissions for the world.

Every file and directory in your UNIX/Linux system has following 3 permissions defined for all the 3 owners discussed above.

* **Read:** This permission give you the authority to open and read a file. Read permission on a directory gives you the ability to lists its content.
* **Write:**The write permission gives you the authority to modify the contents of a file. The write permission on a directory gives you the authority to add, remove and rename files stored in the directory. Consider a scenario where you have to write permission on file but do not have write permission on the directory where the file is stored. You will be able to modify the file contents. But you will not be able to rename, move or remove the file from the directory.
* **Execute:**In Windows, an executable program usually has an extension ".exe" and which you can easily run. In Unix/Linux, you cannot run a program unless the execute permission is set. If the execute permission is not set, you might still be able to see/modify the program code (provided read & write permissions are set), but not run it.





**3. Relative and absolute permission.**

**4. Directory permission.**

**5. Changing file ownership.**

[](https://www.guru99.com/images/its_a_file.png)

**Here, '-rw-rw-r--'**and this weird looking code is the one that tells us about the permissions given to the owner, user group and the world respectively. The first '**-**' implies that we have selected a file.p>

Else, if it were a directory, **d**would have been shown.

[File Permissions in Linux/Unix](https://www.guru99.com/images/Directory.png)

The characters are pretty easy to remember.

**r** = read permission  
**w** = write permission  
**x** = execute permission  
**-** = no permission

Let us look at it this way.

The first part of the code is **'rw-'**. This suggests that the owner 'Home' can:



**No execution permission for owner**

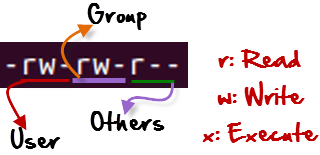
* Read the file
* Write or edit the file
* He cannot execute the file since the execute bit is set to '-'.

The second part is **'rw-'.** It for the user group and group-members can:

* Read the file
* Write or edit the file

The third part is for the world which means any user. It says **'r--'.** This means the user can only:

* Read the file

[](https://www.guru99.com/images/permission(1).png)

**Changing file/directory permissions with 'chmod' command**

Say you do not want your colleague to see your personal images. This can be achieved by changing file permissions.

We can use the 'chmod' command which stands for 'change mode'. Using the command, we can set permissions (read, write, execute) on a file/directory for the owner, group and the world. Syntax:

chmod permissions filename

There are 2 ways to use the command -

1. Absolute mode
2. Symbolic mode

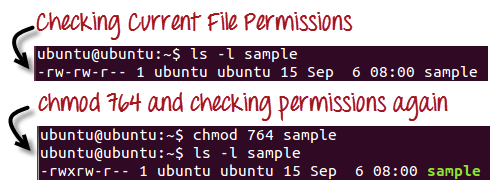
**Absolute(Numeric) Mode**

In this mode, file permissions are not represented as characters but a three-digit octal number.

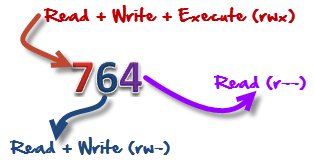
The table below gives numbers for all for permissions types.

|  |  |  |
| --- | --- | --- |
| Number | Permission Type | Symbol |
| 0 | No Permission | --- |
| 1 | Execute | --x |
| 2 | Write | -w- |
| 3 | Execute + Write | -wx |
| 4 | Read | r-- |
| 5 | Read + Execute | r-x |
| 6 | Read +Write | rw- |
| 7 | Read + Write +Execute | rwx |

Let's see the chmod command in action.

[](https://www.guru99.com/images/chmod_new(1).png)

In the above-given terminal window, we have changed the permissions of the file 'sample to '764'.

[](https://www.guru99.com/images/FilePermissions(1).png)

'764' absolute code says the following:

* Owner can read, write and execute
* User-group can read and write
* World can only read

**Symbolic Mode**

In the Absolute mode, you change permissions for all 3 owners. In the symbolic mode, you can modify permissions of a specific owner. It makes use of mathematical symbols to modify the file permissions.

|  |  |
| --- | --- |
| Operator | Description |
| + | Adds a permission to a file or directory |
| - | Removes the permission |
| = | Sets the permission and overrides the permissions set earlier. |

The various owners are represented as -

|  |  |
| --- | --- |
| User Denotations | |
| u | user/owner |
| g | Group |
| o | Other |
| a | All |

We will not be using permissions in numbers like 755 but characters like rwx. Let's look into an example



Initial state



Changing other mode to rwx



Changing other mode to read only

**6. Changing group of a particular file.**

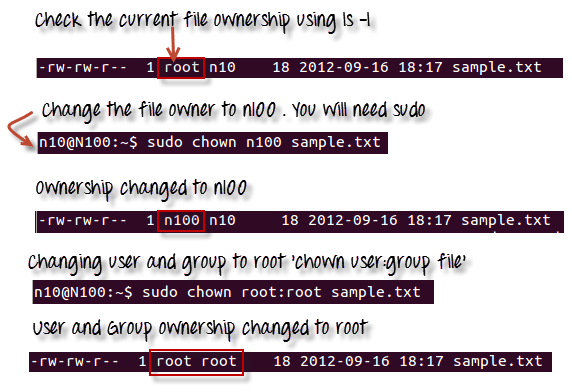
For changing the ownership of a file/directory, you can use the following command:

**chown user**

In case you want to change the user as well as group for a file or directory use the command

**chown user:group filename**

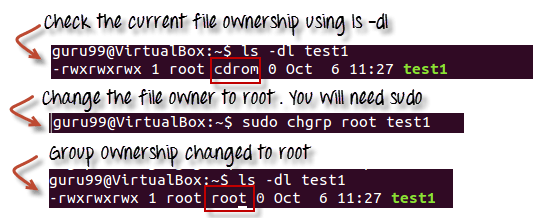
Let's see this in action

[](https://www.guru99.com/images/chown_comm(1).png)

In case you want to change group-owner only, use the command

**chgrp group\_name filename**

'**chgrp'** stands for change group.

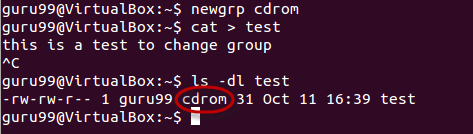
[](https://www.guru99.com/images/chgrp.png)

**Tip**

* The file /etc/group contains all the groups defined in the system
* You can use the command "groups" to find all the groups you are a member of

[File Permissions in Linux/Unix](https://www.guru99.com/images/groups.png)

* You can use the command newgrp to work as a member a group other than your default group

[](https://www.guru99.com/images/newgrp.png)

* You cannot have 2 groups owning the same file.
* You do not have nested groups in Linux. One group cannot be sub-group of other
* x- eXecuting a directory means Being allowed to "enter" a dir and gain possible access to sub-dirs
* There are other permissions that you can set on Files and Directories which will be covered in a later advanced tutorial

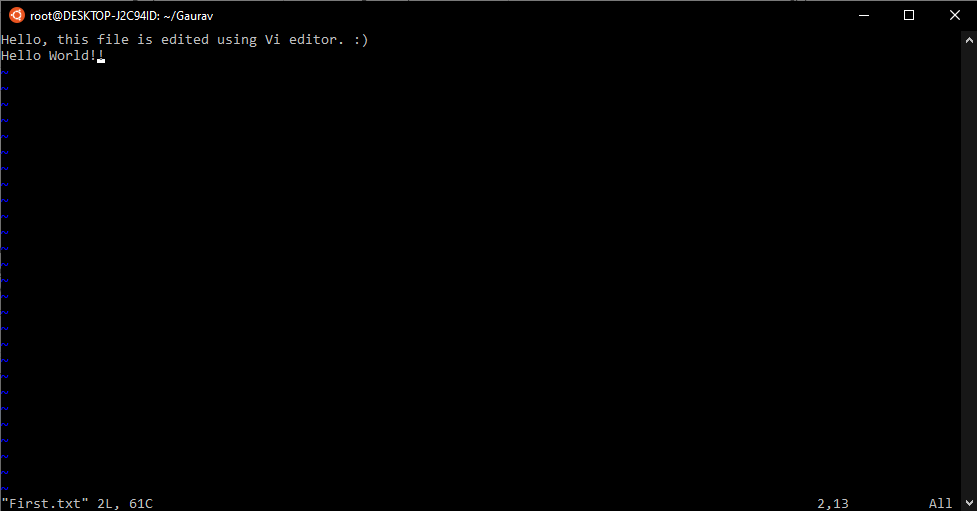
## Part E:

**1. vi Editor and its basics.**

**-> vi** editor is used to create, edit files in shell interface. We can open editor mode by writing:

**vi** <NEW\_FILE> / <EXISTING\_FILE> and press enter.

The interface is very simple as shown:



In this window, the existing file is opened, so the text is shown, on the left ~ indicates empty lines.

To edit or to write in this file, type **i/a** to input the text. After completion, press esc to exit the input mode. In bottom the status of file is shown, to write the input, press **:** and **w** for writing on the file and to quit the editor enter **:q**

To check the contents of file, use the command **cat**.



**2. Repeat factor.**

vi mode commands may be prefixed by a command repeat factor. This means that most character, change, delete, word, movement, and positioning commands may be preceded by a number which refers to the number of times the command should be repeated. For example, **j** moves the cursor 1 character down, and **4j** moves the cursor 4 characters down. Here are various commands:

|  |  |
| --- | --- |
| <Esc> | Return to command mode. |
| <Ret> or + | Go to the first non-blank character of the next line. |
| - | Go to the first non-blank character of the previous line (**-** is the minus sign). |
| A | Append text after the cursor. |
| A | Append text at the end of the current line. |
| Dd | Delete the current line. |
| D or d$ | Delete from the cursor to the end of the line. |
| U | Undo the last change. |
| U | Undo all changes to the current line. |
| X | Delete the current character (the one at the cursor). |
| *N*x | Delete *n* characters to the right, starting with the current character. |
| ZZ | Save changes and exit. |
| :wq<Ret> | Save changes and exit. |
| :w<Ret> | Save changes without exiting. |
| :q!<Ret> | Quit without saving changes. |

**3. Input mode and insertion of text.**

As explained in the basics, the other various options are:

|  |  |
| --- | --- |
| a | Append text after the cursor. |
| *n*a | Append after the cursor. Whatever you type until you press **<Esc>**gets replicated *n* times. For example: if you type **3a**, then **12<Esc>**, the string **121212** is inserted. |
| A | Append text at the end of the current line. |
| i | Insert text before the cursor. |
| I | Insert text at the beginning of the current line. |
| o | Open a new line below the current line. |
| O | Open a new line above the current line. |

**4. Saving text and quitting.**

|  |  |
| --- | --- |
| ZZ | Save changes and exit. |
| :wq<Ret> | Save changes and exit. |
| :w<Ret> | Save changes without exiting. |
| :q!<Ret> | Quit without saving changes. |

**5. Navigation.**

For navigations there are many many options:

**<Ctrl>** represents the Control key. **<Ctrl>f** means hold the Control key, then press the f key.

|  |  |
| --- | --- |
| <Ctrl>&f | Move forward (downward) by 1 screen. |
| <Ctrl>b | Move backward (upward) by 1 screen. |
| <Ctrl>d | Move downward by 1/2 screen. |
| <Ctrl>u | Move upward by 1/2 screen. |
| <Ctrl>e | Scroll downward one line. |
| <Ctrl>y | Scroll upward one line. |
| H | Go to the first line on the screen. |
| M | Go to the middle line on the screen. |
| L | Go to the last line on the screen. |
| z<Ret> | Shifts the current line to the top of the screen. |
| z- | Shifts the current line to the bottom of the screen. |
| z. | Shifts the current line to the center of the screen. |

**6. Editing text.**

In vi, a sentence ends with a **.**, **!**, or **?**, followed by **TWO BLANK SPACES** (one blank space doesn't cut it!) or a newline (what you get when you type **<Ret>**). Paragraphs are separated by one or more blank lines.

|  |  |
| --- | --- |
| ) | Move to the next sentence. |
| ( | Move to the beginning of the current sentence, (or to the beginning of the previous sentence if you are already at the beginning of a sentence). |
| } | Move to next paragraph. |
| { | Move to the end of the previous paragraph. |

**Shifting Text**

|  |  |
| --- | --- |
| >> | Shift right one tab stop. |
| n>> | Shift n lines right one tab stop. |
| << | Shift left one tab stop. |
| n<< | Shift n lines left one tab stop. |

Note: The default tabstop is 8 characters. To change your tabstop, use the **:set tabstop** command, as explained below in the section called "The .exrc vi Startup File and Setting vi Parameters".

**Deletion Commands**

|  |  |
| --- | --- |
| X | Delete the current character (the one at the cursor). |
| nx | Delete n characters to the right, starting with the current character. |
| X | Delete the character to the left of the cursor. |
| nX | Delete n characters to the left of the cursor (starting with the character to the left of the cursor). |
| dd | Delete the current line. |
| dw | Delete from the cursor to end of the current word. |
| db | Delete from the character to the left of the cursor leftward to the start of the current word. |
| dL | Delete all lines from the current line downward till the bottom of the screen. |
| dH | Delete all lines from the current line upward till the top of the screen. |
| dM | Delete lines from the current line to the middle of the screen (either direction depending on current location). |
| dG | Delete from the current line through the end of the file. |

**7. Undo last editing.**

|  |  |
| --- | --- |
| u | Undo the last change. |
| U | Undo all changes to the current line. |

**8. Repeating last command.**

**9. Searching for a pattern.**

**Simple Search**

The slash (**/**) is used to search forwards, and the question mark (**?**) to search backwards. Both **/** and **?** are last line mode commands: when you type a **/** or a **?** your cursor moves to the last line of the screen, where you type the rest of the search command. Search commands are executed only after you press **<Ret>**. If you want to exit last line mode without executing the search command, type **<Ctrl>c**.

Case is respected in searches unless you issue the ex command: **:set ignorecase<Ret>**

which can be abbreviated: **:set ic<Ret>**

The search will wrap around the end-of-file to the beginning-of-file or vice versa, depending on the direction of the search, unless you issue the ex command: **:set nowrapscan<Ret>**

which can be abbreviated: **:set nows<Ret>**

in which case the search will stop when it reaches the last line in its search direction.

To repeat the search in its original direction, type: **n**

To reverse the direction of the original search, type: **N**

Example of a forward search on the word greetings: **/greetings<Ret>**

If the string is found, to search backward for the next occurrence, type: **N**

There are two global search commands, one (**g**) which includes the search string, and one (**v**) which excludes the search string. These search commands may be followed by edit commands, such as **d** to delete the line in which the search string appears or doesn't appear. For example:

**:g/gimbel/d<Ret>** Deletes all lines that contain the string **gimbel**.

**:v/gyre/d<Ret>** Deletes all lines that DO NOT contain the string **gyre**.

**10. Substitution – search and replace.**

The search and substitute command has two option flags: the **g**flag for making global changes, and the **c** flag for making conditional changes (these are described below). The format of the search and substitute command is: **:[address]s/search\_string/replace\_string/flags<Ret>**

To search on the current line for the first occurrence of "may" and replace it by the word "will": **:s/may/will<Ret>**

To change all of the occurrences of "may" to "will" on the current line, use the **g** (for global) flag: **:s/may/will/g<Ret>**

In the following example, both the **g** and **c**flags are used. The **g** means global, and will cause all occurrences in lines 1 through 5 to be selected, instead of just the first occurrence in each line. The **c**means conditional, and means that you will be asked whether or not to make the substitution. **:1,5s/want/would like/cg<Ret>**

Respond: **y<Ret>**

for yes, or: **n<Ret>**

for no (the default).

To replace all the words "blue" with the word "red" throughout the file: **:%s/blue/red/g<Ret>**

The ampersand (**&**) is a special "metacharacter". If it is used in the replacement string, it keeps the search string as part of the replacement. For example, to change "red" to "red car": **:s/red/& car/<Ret>**

# **Lab 3**

1. **Write a program to implement a First-Come-First-Serve Scheduling algorithm.**

**>|** #include<iostream>

using namespace std;

int Insert(string q[],int t[],int cnt){

    cin>>q[cnt]>>t[cnt];

    cnt++;

    return cnt;

}

int Delete(string q[],int t[],int cnt){

    for(int i=0;i<cnt;i++){

        q[i] = q[i+1];

        t[i] = t[i+1];}

    cnt--;

    return cnt;

}

void Print\_all\_jobs(string q[],int t[],int cnt){

for(int i=0;i<cnt;i++)

    cout<<q[i]<<" :"<<t[i]<<" |";

cout<<endl;

}

int main(){

    cout<<"The First-Come-First-Serve(FCFS)Scheduling:"<<endl;

    int time[10];

    int cnt=0;

    string job\_name[10];

    while(1){

    int c;

    cout<<"Options:1)Insert\t2)Delete\t3)Print All\t4)Exit:";

    cin>>c;

    switch(c){

    case 1:cnt=Insert(job\_name,time,cnt);

    break;

    case 2:cnt = Delete(job\_name,time,cnt);

    break;

    case 3:Print\_all\_jobs(job\_name,time,cnt);

    break;

    case 4:return 0;

    break;

    default:cout<<"Enter 1/2/3/4 please:";

    break;

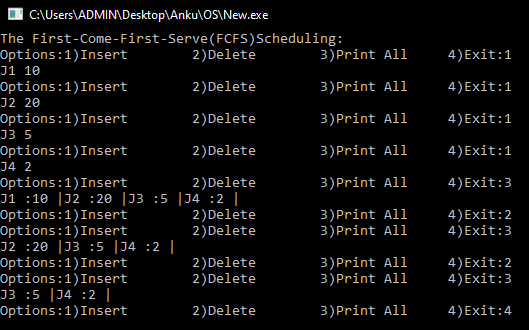
    }

    }

    return 0;

}

**Output:**

****

1. **Write a program to implement Shortest Job First Scheduling algorithm.**

**>|** #include<iostream>

using namespace std;

void refactor(string q[],int t[],int cnt){

    for(int i=0;i<cnt-1;i++){

        if(t[cnt-1] < t[i]){

                int temp\_t = t[cnt-1];

                string temp\_q =q[cnt-1];

            for(int j=cnt-1;j>=i;j--){

                t[j+1] = t[j];

                q[j+1]= q[j];

            }

            t[i] = temp\_t;

            q[i] = temp\_q;

        }

    }

}

int Insert(string q[],int t[],int cnt){

    cin>>q[cnt]>>t[cnt];

    cnt++;

    refactor(q,t,cnt);

    return cnt;

}

int Delete(string q[],int t[],int cnt){

    for(int i=0;i<cnt;i++){

        q[i] = q[i+1];

        t[i] = t[i+1];}

    cnt--;

    return cnt;

}

void Print\_all\_jobs(string q[],int t[],int cnt){

for(int i=0;i<cnt;i++)

    cout<<q[i]<<" :"<<t[i]<<" |";

cout<<endl;

}

int main(){

    cout<<"The Shortest-Job-First(SJF) Scheduling:"<<endl;

    int time[10];

    int cnt=0;

    string job\_name[10];

    //char s[];

    while(1){

    int c;

    cout<<"Options:1)Insert\t2)Delete\t3)Print All\t4)Exit:";

    cin>>c;

    switch(c){

    case 1:cnt=Insert(job\_name,time,cnt);

    break;

    case 2:cnt = Delete(job\_name,time,cnt);

    break;

    case 3:Print\_all\_jobs(job\_name,time,cnt);

    break;

    case 4:return 0;

    break;

    default:cout<<"Enter 1/2/3/4 please:";

    break;

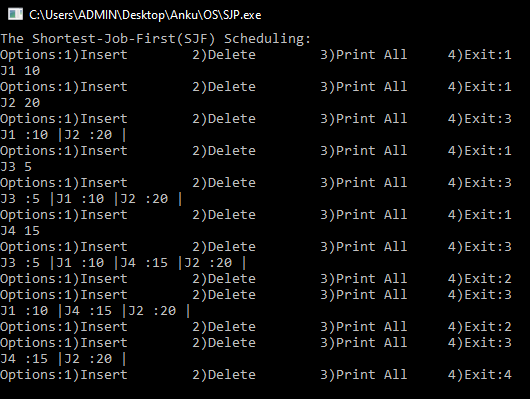
    }

    }

    return 0;

}

**Output:**

****

1. **Write a program to implement Shortest Remaining Time First Scheduling algorithm:**

**>|** #include<iostream>

using namespace std;

void refactor(string q[],int At[],int bt[],int cnt){

    for(int i=0;i<cnt-1;i++){

        if((bt[cnt-1]-At[cnt-1]) < (bt[i]-At[i])){

                int temp\_at = At[cnt-1];

                int tem\_bt=bt[cnt-1];

                string temp\_q =q[cnt-1];

            for(int j=cnt-1;j>=i;j--){

                At[j+1] = At[j];

                bt[j+1]=bt[j];

                q[j+1]= q[j];

            }

            At[i] = temp\_at;

            bt[i]=tem\_bt;

            q[i] = temp\_q;

        }

    }

}

int Insert(string q[],int At[],int bt[],int cnt){

    cin>>q[cnt]>>At[cnt]>>bt[cnt];

    cnt++;

    if((bt[cnt-1]-At[cnt-1]) < (bt[0]-At[0]))

        bt[0] -= (At[cnt-1]-At[0]);

    refactor(q,At,bt,cnt);

    return cnt;

}

int Delete(string q[],int At[],int bt[],int cnt){

    for(int i=0;i<cnt;i++){

        q[i] = q[i+1];

        At[i] = At[i+1];

        bt[i] = bt[i+1];

        }

    cnt--;

    return cnt;

}

void Print\_all\_jobs(string q[],int At[],int bt[],int cnt){

for(int i=0;i<cnt;i++)

    cout<<q[i]<<" |\t"<<At[i]<<" |\t"<<bt[i]<<" |\n";

cout<<endl;

}

int main(){

    cout<<"The Shortest-Remaining-Time-First(SRTF) Scheduling:"<<endl;

    int Arrival[10],Burst[10];

    int cnt=0;

    string job\_name[10];

    while(1){

        int c;

        cout<<"Options:1)Insert\t2)Delete\t3)Print All\t4)Exit:";

        cin>>c;

        switch(c){

            case 1:cnt=Insert(job\_name,Arrival,Burst,cnt);

            break;

            case 2:cnt = Delete(job\_name,Arrival,Burst,cnt);

            break;

            case 3:Print\_all\_jobs(job\_name,Arrival,Burst,cnt);

            break;

            case 4:return 0;

            break;

            default:cout<<"Enter 1/2/3/4 please:"<<endl;

            break;

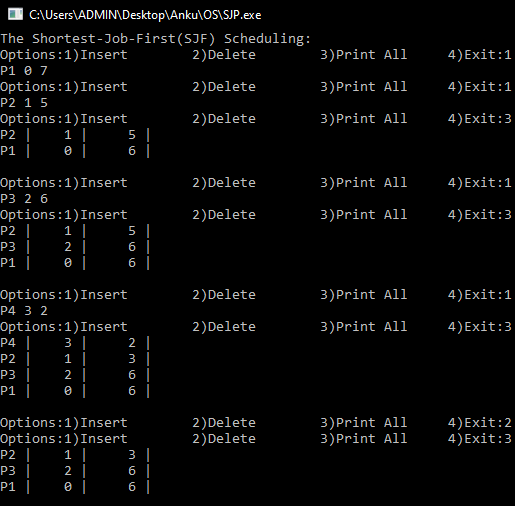
        }

    }

    return 0;

}

**Output:**

****

1. **Write a program to implement Priority (non pre-emptives) Scheduling algorithm:**

#include<iostream>

using namespace std;

void refactor(string q[],int AT[],int bt[],int pri[],int cnt){

**/\*why not start with the 0th element? :- It was first to enter**

**and since the scheduling is non preemptive so Ist process**

**will execute its whole process, rearranging will be done**

**after his element.\*/**

    for(int i=1;i<cnt-1;i++){

        if(pri[cnt-1] < pri[i]){

                int temp\_at = AT[cnt-1];

                string temp\_q =q[cnt-1];

                int temp\_bt = bt[cnt-1];

                int temp\_p =pri[cnt-1];

            for(int j=cnt-1;j>=i;j--){

                AT[j+1] = AT[j];

                q[j+1]= q[j];

                bt[j+1] = bt[j];

                pri[j+1]= pri[j];

            }

            AT[i] = temp\_at;

            q[i] = temp\_q;

            bt[i] = temp\_bt;

            pri[i] = temp\_p;

        }

    }

}

int Insert(string q[],int At[],int bt[],int pri[],int cnt){

    if(cnt<10){

        cnt++;

        cout<<"Enter job-Arrival-burst-priority:"<<endl;

        cin>>q[cnt-1]>>At[cnt-1]>>bt[cnt-1]>>pri[cnt-1];

        refactor(q,At,bt,pri,cnt);

    }

    else{

        cout<<"Queue size FULL"<<endl;

    }

    return cnt;

}

int Delete(string q[],int At[],int bt[],int pri[],int n){

    if(n > 0){

        for(int j=0;j<n-1;j++){

            q[j]= q[j+1];

            At[j]= At[j+1];

            bt[j]=bt[j+1];

            pri[j]=pri[j+1];

        }

        return (n-1);

    }

    else{

        cout<<"Queue Size 0"<<endl;

    return n;

    }

}

void Print\_all\_jobs(string q[],int At[],int bt[],int pri[],int cnt){

    cout<<"Job |\tArrival |\tBurst |\tPriority |"<<endl;

    for(int i=0;i<cnt;i++)

        cout<<q[i]<<" |\t"<<At[i]<<" |\t"<<bt[i]<<" |\t"<<pri[i]<<" |\n";

    cout<<endl;

}

int main(){

    cout<<"The Priority\_based(P- np) Scheduling:"<<endl;

    int A\_t[10],B\_t[10],prir[10];

    int cnt=0;

    string job\_name[10];

    while(1){

        int c;

        cout<<"Options:1)Insert\t2)Delete\t3)Print All\t4)Exit:";

        cin>>c;

        switch(c){

            case 1:cnt=Insert(job\_name,A\_t,B\_t,prir,cnt);

            break;

            case 2:cnt=Delete(job\_name,A\_t,B\_t,prir,cnt);

            break;

            case 3:Print\_all\_jobs(job\_name,A\_t,B\_t,prir,cnt);

            break;

            case 4:return 0;

            break;

            default:cout<<"Enter 1/2/3/4 please:";

            break;

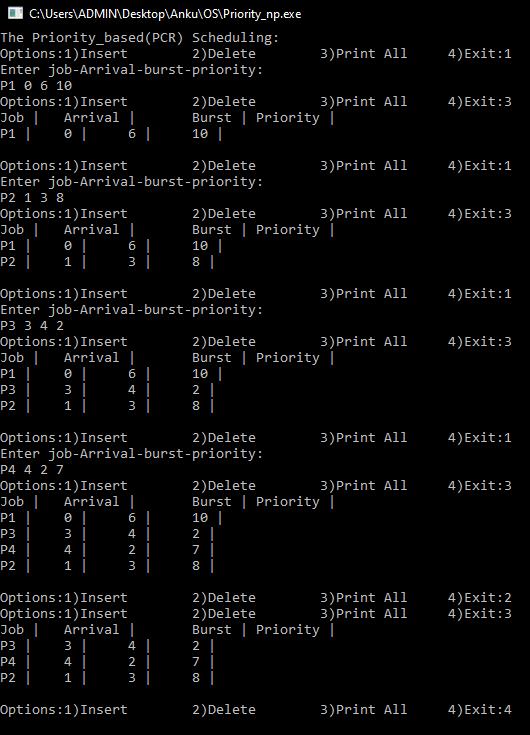
        }

    }

    return 0;

}

**Output:**

****

1. **Write a program to implement Priority (pre-emptive) Scheduling algorithm:**

#include<iostream>

using namespace std;

void refactor(string q[],int AT[],int bt[],int pri[],int cnt){

    for(int i=0;i<cnt-1;i++){

        if(pri[cnt-1] < pri[i]){

                int temp\_at = AT[cnt-1];

                string temp\_q =q[cnt-1];

                int temp\_bt = bt[cnt-1];

                int temp\_p =pri[cnt-1];

            for(int j=cnt-1;j>=i;j--){

                AT[j+1] = AT[j];

                q[j+1]= q[j];

                bt[j+1] = bt[j];

                pri[j+1]= pri[j];

            }

            AT[i] = temp\_at;

            q[i] = temp\_q;

            bt[i] = temp\_bt;

            pri[i] = temp\_p;

        }

    }

}

int Insert(string q[],int At[],int bt[],int pri[],int cnt){

    if(cnt<10){

        cnt++;

        cout<<"Enter job-Arrival-burst-priority:"<<endl;

        cin>>q[cnt-1]>>At[cnt-1]>>bt[cnt-1]>>pri[cnt-1];

        if(pri[cnt-1] < pri[0])

            bt[0] -= (At[cnt-1]-At[0]);

        refactor(q,At,bt,pri,cnt);

    }

    else{

        cout<<"Queue size FULL"<<endl;

    }

    return cnt;

}

int Delete(string q[],int At[],int bt[],int pri[],int n){

    if(n > 0){

        for(int j=0;j<n-1;j++){

            q[j]= q[j+1];

            At[j]= At[j+1];

            bt[j]=bt[j+1];

            pri[j]=pri[j+1];

        }

        return (n-1);

    }

    else{

        cout<<"Queue Size 0"<<endl;

    return n;

    }

}

void Print\_all\_jobs(string q[],int At[],int bt[],int pri[],int cnt){

    cout<<"Job |\tArrival |\tBurst |\tPriority |"<<endl;

    for(int i=0;i<cnt;i++)

        cout<<q[i]<<" |\t"<<At[i]<<" |\t"<<bt[i]<<" |\t"<<pri[i]<<" |\n";

    cout<<endl;

}

int main(){

    cout<<"The Priority\_based(P- p) Scheduling:"<<endl;

    int A\_t[10],B\_t[10],prir[10];

    int cnt=0;

    string job\_name[10];

    while(1){

        int c;

        cout<<"Options:1)Insert\t2)Delete\t3)Print All\t4)Exit:";

        cin>>c;

        switch(c){

            case 1:cnt=Insert(job\_name,A\_t,B\_t,prir,cnt);

            break;

            case 2:cnt=Delete(job\_name,A\_t,B\_t,prir,cnt);

            break;

            case 3:Print\_all\_jobs(job\_name,A\_t,B\_t,prir,cnt);

            break;

            case 4:return 0;

            break;

            default:cout<<"Enter 1/2/3/4 please:";

            break;

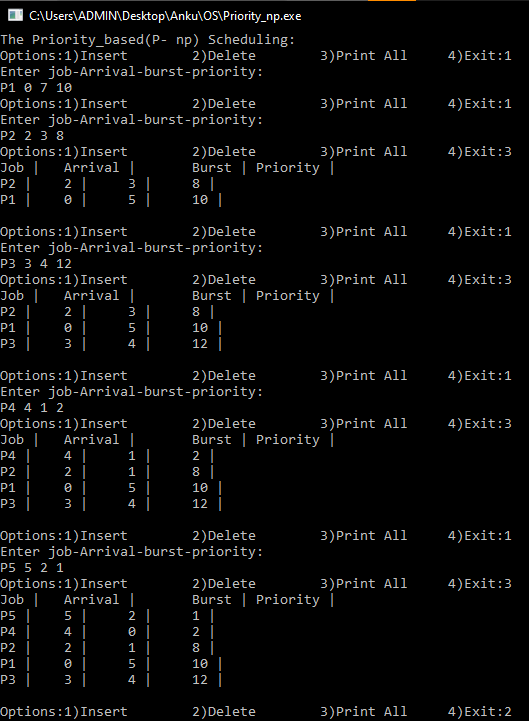
        }

    }

    return 0;

}

**Output:**

****

1. **Write a program to implement Round Robin Scheduling algorithm:**

**>|** #include<iostream>

#include<queue>

using namespace std;

int Delete(string q[],int At[],int bt[],int ind,int n){

    for(int i=0;i<n-1;i++){

        q[i]=q[i+1];

        At[i]=At[i+1];

        bt[i]=bt[i+1];

    }

    return (n-1);

}

int main(){

    queue<string> Ready;

    queue<string> Running;

    int n=5,T=2;

    int A\_T[10],B\_T[10];

    string process[10];

    for(int i=0;i<n;i++){

        cin>>process[i]>>A\_T[i]>>B\_T[i];

        Ready.push(process[i]);

    }

    int i=0;

    while(!Ready.empty()){

        string prcs = Ready.front();

        Ready.pop();

        Running.push(prcs);

        if((B\_T[i] - T) <= 0)

            n= Delete(process,A\_T,B\_T[],i,n);

        else

            Ready.push(prcs);

        B\_T[i] -= T;

        i=(i+1)%n;

    }

    while(!Running.empty()){

        cout<<Running.front()<<" |";

        Running.pop();

    }

    return 0;

}

**Output:**

# **Lab 4**

**Producer consumer Problem using semaphores:**

#include <iostream>

using namespace std;

#define N 10

typedef int semaphore;

semaphore mutex=1,empty=N,full=0;

int buff[N];

void down(int \*c){

    if (\*c > 0)

        \*c -= 1;

}

void up(int \*c){

    if (\*c < N)

        \*c =(\*c +1)%N;

}

void produce\_item(int \*item){

    int it;

    cin>>it;

    \*item = it;

    cout<<"Produced......OK"<<endl;

}

void enter\_item(int item){

    buff[full] = item;

}

void consume\_item(int item){

    cout<<item<<" Consumed......OK"<<endl;

}

void remove\_item(int \*item){

    \*item=buff[empty];

}

void print(){

    for(int i=0;i<N;i++)

        cout<<buff[i]<<" ";

    cout<<endl;

}

void producer(){

    int item;

    while(empty > 0 && mutex==1){

        produce\_item(&item);

        down(&empty);

        down(&mutex);

        enter\_item(item);

        up(&mutex);

        up(&full);

    }

}

void consumer(){

    int item;

    while(empty < N && mutex==1){

        down(&full);

        down(&mutex);

        remove\_item(&item);

        up(&mutex);

        up(&empty);

        consume\_item(item);

    }

}

int main(){  cout<<"============>Producer-Consumer-Problem-with-Semaphore<============"<<endl;

    while(1){

        int ch;

        cout<<"1. Producer\t 2. Consumer\t 3. Buffer\t 4. Exit:";

        cin>>ch;

        switch(ch){

        case 1: producer();

        break;

        case 2: consumer();

        break;

        case 3: print();

        break;

        case 4: return 0;

        break;

        default:cout<<"Enter 1/2/3/4 +->"<<endl;

        break;

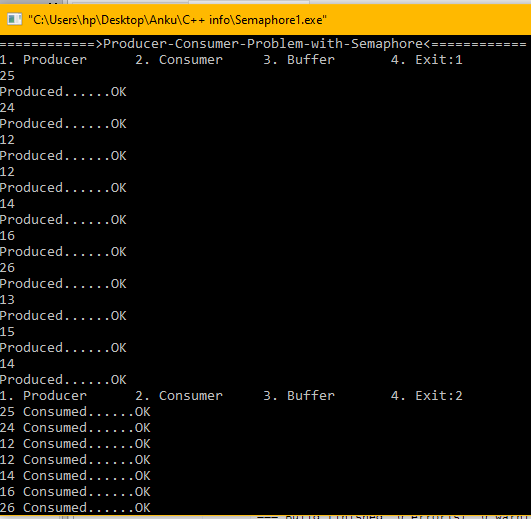
        }

    }

    return 0;

}

**Output:**

****

**Reader Writer Problem using Semaphores:**

#include <iostream>

using namespace std;

#define N 10

typedef int semaphore;

semaphore mutex=1,db=1;

int database[10] = {0,1,2,3,4,5,6,7,8,9};

int rc=0,wc=0;

void down(int \*c){

    if (\*c > 0)

        \*c -= 1;

}

void up(int \*c){

    if (\*c < N)

        \*c =(\*c +1)%N;

}

void print(){

    cout<<"Data : ";

    for(int i=0;i<N;i++)

        cout<<database[i]<<" ";

    cout<<endl;

    cout<<"current readers: "<<rc<<endl;

}

void reader(){

//    while(true){

        int ch;

        cout<<"1. In \t 2. Out:"<<endl;

        cin>>ch;

        down(&mutex);

        if (ch == 1 && wc == 0){

            rc += 1;

            if(rc == 1)

                down(&db);

            up(&mutex);

            //read database

            int i;

            cin>>i;

            int item = database[i];

            down(&mutex);

            //use data

            cout<<"Data: "<<item<<endl;

        }

        else if(ch == 2)

            rc -= 1;

        else if(wc != 0)

            cout<<"A writer is existing..."<<endl;

        cout<<"Reader count:"<<rc<<endl;

        if(rc == 0)

            up(&db);

        up(&mutex);

}

void writer(){

    //while(empty != 0/\*true\*/){

        down(&db);

        int ch;

        cout<<"1. In \t 2. Out:"<<endl;

        cin>>ch;

        if(rc == 0 && wc == 0 && ch == 1){ // write only when no readers/writers are there

        //write data to database

            wc = 1;

            int i,item;

            cin>>i>>item;

            database[i]= item;

            up(&db);

        }

        else if(ch == 2)

            wc = 0;

        else

            cout<<"A reader/writer is existing..."<<endl;

    //}

}

int main(){   cout<<"============>Reader-Writer-Problem-with-Semaphore<============"<<endl;

    while(1){

        int ch;

        cout<<"1. Reader\t 2. Writer\t 3. Buffer\t 4. Exit:";

        cin>>ch;

        switch(ch){

        case 1: reader();

        break;

        case 2: writer();

        break;

        case 3: print();

        break;

        case 4: return 0;

        break;

        default:cout<<"Enter 1/2/3/4 +->"<<endl;

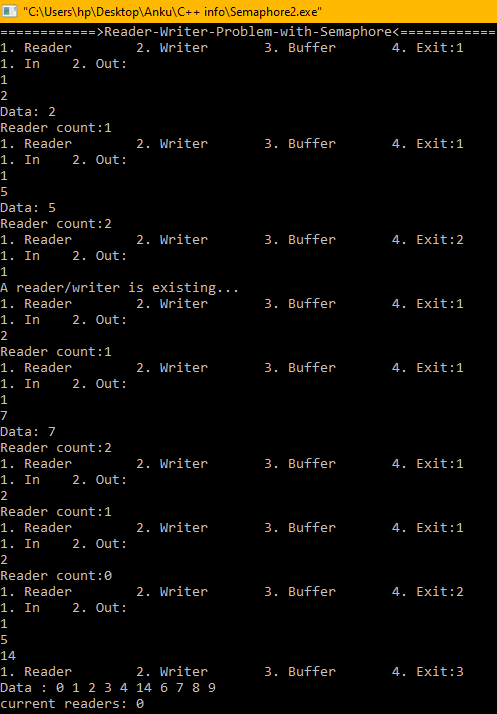
        break;

        }

    }

    return 0;

}

****

**Dining Philosopher Problem using Semaphores:**

#include<iostream>

#include<thread>

#include<chrono>

#define N 5

using namespace std;

typedef int semaphore;

int state[N];

semaphore mutex=1;

semaphore s[N];

int a=0;

int left(int i){

    if(i == 0)

        i = N-1;

    else

        i = (i-1)%N;

    return i;

}

int right(int i){

    i = (i+1)%N;

    return i;

}

void test(int i){

    if(state[i] == 1 && state[left(i)] != 2 && state[right(i)] != 2){

        state[i] = 2;

        s[i] += 1; }

}

void think(){

    cout<<a<<"  Thinking....."<<endl;

}

void eat(){

    cout<<a<<"  Eating.....Yummy! :-) "<<endl;

    this\_thread::sleep\_for(chrono::seconds(2));

}

void take\_fork(int i){

    mutex = 0;

    state[i] = 1;

    test(i);

    mutex = 1;

    s[i] -= 1;

}

void put\_fork(int i){

    mutex = 0;

    state[i] = 0;

    test(left(i));

    test(right(i));

    mutex = 1;

}

void philosopher(int i){

    think();

    take\_fork(i);

    a=i;

    eat();

    put\_fork(i);

}

int main(){

    cout<<"======>Dining-Philosophers-Problem-with-Semaphores<======"<<endl;

    thread T1(philosopher,0);

    thread T2(philosopher,1);

    thread T3(philosopher,2);

    thread T4(philosopher,3);

    thread T5(philosopher,4);

    T1.join();

    T2.join();

    T3.join();

    T4.join();

    T5.join();

    return 0; }

**Output:**

****

# **Lab 5**

**Implementing Banker’s Algorithm:**

#include<iostream>

#define N 3

#define M 5

using namespace std;

void Print(int alloc[M][N],int Mx\_nd[M][N],int rsrc[N],int rmsrc[N],int rm\_nd[M][N]){

    cout<<"   |\tAllocation | \t Maximum Need   | \t Remaining |"<<endl;

    for(int i=0;i<M;i++){

        cout<<"P"<<i+1<<" | \t";

        for(int j=0;j<N;j++)

            cout<<alloc[i][j]<<" ";

        cout<<" \t   | \t ";

        for(int k=0;k<N;k++)

            cout<<Mx\_nd[i][k]<<" ";

        cout<<" \t| \t ";

        for(int j=0;j<N;j++)

            cout<<rm\_nd[i][j]<<" ";

        cout<<" |"<<endl;

    }

    cout<<endl;

    cout<<"Available Resources: |";

    for(int i=0;i<N;i++)

        cout<<rsrc[i]<<" ";

    cout<<" |  \t Remaining Resources: |";

    for(int j=0;j<N;j++)

            cout<<rmsrc[j]<<" ";

    cout<<" |"<<endl;

}

void Banker(int alloc[M][N],int Mx\_nd[M][N],int rsrc[N]){

    //Remaining Sources

    int Remain\_resources[N]={0,0,0};

    for(int i=0;i<M;i++)

        for(int j=0;j<N;j++)

            Remain\_resources[j] += alloc[i][j];

    for(int i=0;i<N;i++)

        Remain\_resources[i] = rsrc[i]-Remain\_resources[i];

    //Remaining Needs

    int Remain\_need[M][N];

    for(int i=0;i<M;i++)

        for(int j=0;j<N;j++)

            Remain\_need[i][j] = Mx\_nd[i][j] - alloc[i][j];

    //Process

    int t=0,cnt=0,pcnt =0;

    int Process[M];

    l1:for(int i=0;i<M;i++){

        int flag=0;

        for(int j=0;j<N;j++){

            if(Remain\_need[i][j] <= Remain\_resources[j] && Remain\_need[i][j] != -1)

                flag += 1;

        }

        if(flag == N){

            for(int k=0;k<N;k++){

                    Remain\_resources[k] += alloc[i][k];

                    Remain\_need[i][k] = -1;

                    Process[t] = i+1;

                }

                t += 1;

                cnt += 1;

        }

        Print(alloc,Mx\_nd,rsrc,Remain\_resources,Remain\_need);

        flag=0;

    }

    pcnt += 1;

    if(cnt == 0)

        cout<<"Unsafe Deadlock condition"<<endl;

    if(pcnt > 5)

        cout<<"Deadlock state"<<endl;

    if(cnt != M)

        goto l1;

    cout<<endl<<"Process Execution: ";

    for(int i=0;i<M;i++)

        cout<<"P"<<Process[i]<<" ";

    cout<<endl;

}

int main(){

    int resource[N];//={10,5,7};

    int Alloc[M][N];

**/\*={{0,1,0},**

**{2,0,0},**

**{3,0,2},**

**{2,1,1},**

**{0,0,2}};\*/**

    int Max\_need[M][N];

**/\*={{7,5,3},**

**{3,2,2},**

**{9,0,2},**

**{4,2,2},**

**{5,3,3}};\*/**

    cout<<"Enter the resources:"<<endl;

    for(int i=0;i<N;i++)

        cin>>resource[i];

    cout<<"Enter no. of resources allocated to process:"<<endl;

    for(int i=0;i<M;i++)

        for(int j=0;j<N;j++)

            cin>>Alloc[i][j];

    cout<<"Enter maximum need:"<<endl;

    for(int i=0;i<M;i++)

        for(int j=0;j<N;j++)

            cin>>Max\_need[i][j];

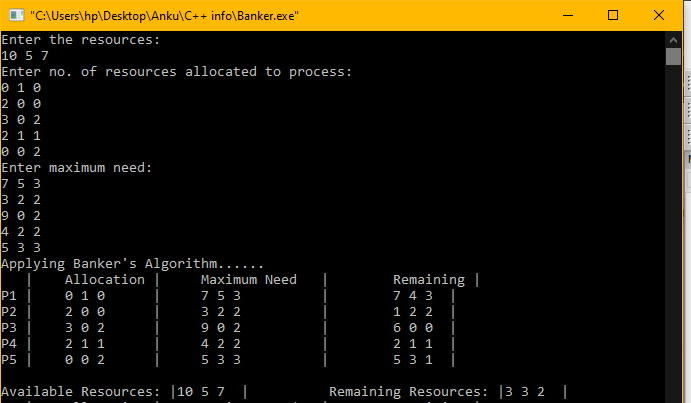
    cout<<"Applying Banker's Algorithm......"<<endl;

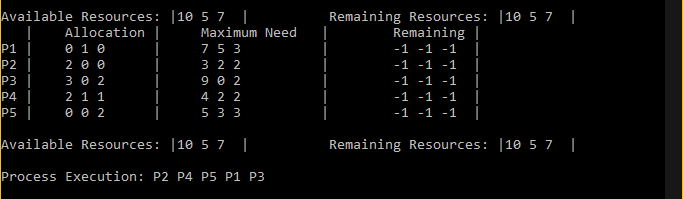
    Banker(Alloc,Max\_need,resource);

    return 0;

}

**Output:**

****

****

# **Lab 6**

**Q.Write a C program to simulate following Memory Allocation Strategies.**

**(a) First Fit Allocation**

**(b) Best Fit Allocation**

**(c) Worst Fit Allocation**

**Ans.**

#include <iostream>

using namespace std;

void print(int p[],int n){

    for(int i=0;i<n;i++)

        cout<<p[i]<<" |";

    cout<<endl;

}

void **First\_Fit**(int mem[],int p[],int mem\_size,int p\_size){

    cout<<"      First-Fit      "<<endl;

    // copying into temporary array

    int temp[mem\_size];

    for(int i=0;i<mem\_size;i++)

        temp[i] = mem[i];

    int tempp[p\_size];

    for(int i=0;i<p\_size;i++)

        tempp[i] = p[i];

    for(int i=0;i<p\_size;i++){

        for(int j=0;j<mem\_size;j++){

            if(temp[j] >= tempp[i] && temp[j] != 0){

                temp[j] -= tempp[i];

                tempp[i] = j;

                break;

            }

        }

    }

    cout<<"Memory: \t";

    print(temp,mem\_size);

    cout<<"Processes: \t";

    print(tempp,p\_size);

}

void **Next\_Fit**(int mem[],int p[],int mem\_size,int p\_size){

    cout<<"      Next-Fit      "<<endl;

    // copying into temporary memory

    int temp[mem\_size];

    for(int i=0;i<mem\_size;i++)

        temp[i] = mem[i];

    // copying into temporary process array

    int tempp[p\_size];

    for(int i=0;i<p\_size;i++)

        tempp[i] = p[i];

    int k=0;

    for(int i=0;i<p\_size;i++){

        for(int j=k;j<mem\_size;j++){

            if(temp[j] >= tempp[i] && temp[j] != 0){

                temp[j] -= tempp[i];

                tempp[i] = j;

                k=j;

                break;

            }

        }

    }

    cout<<"Memory: \t";

    print(temp,mem\_size);

    cout<<"Processes: \t";

    print(tempp,p\_size);

}

int smallest(int mem[],int m,int p){

    int mn=0,k=0;

    int Tm[m];

    for(int i=0;i<m;i++){

        if(mem[i] != 0){

            Tm[k] = mem[i];

            k++;

        }

    }

    //sort the memory

    for(int i=0;i<k;i++){

        for(int j=0;j<k;j++){

            if(Tm[i]<Tm[j]){

                int t=Tm[i];

                Tm[i]=Tm[j];

                Tm[j]=t;

            }

        }

    }

    for(int i=0;i<k;i++)

        if(Tm[i] >= p){

            mn=Tm[i];

            break;

        }

    return mn;

}

void **Best\_Fit**(int mem[],int p[],int mem\_size,int p\_size){

    cout<<"      Best-Fit      "<<endl;

    // copying into temporary memory

    int temp[mem\_size];

    for(int i=0;i<mem\_size;i++)

        temp[i] = mem[i];

    // copying into temporary process

    int tempp[p\_size];

    for(int i=0;i<p\_size;i++)

        tempp[i] = p[i];

    // Select the smallest possible space enough for process

    // throughout the array.

    for(int i=0;i<p\_size;i++){

        int mn=smallest(temp,mem\_size,tempp[i]);

        for(int j=0;j<mem\_size;j++){

            if(mn == temp[j])

                mn =j;

        }

        temp[mn] -= tempp[i];

        tempp[i] = mn;

    }

    cout<<"Memory: \t";

    print(temp,mem\_size);

    cout<<"Processes: \t";

    print(tempp,p\_size);

}

void **Worst\_Fit**(int mem[],int p[],int mem\_size,int p\_size){

    cout<<"      Worst-Fit      "<<endl;

    // copying into temporary array

    int temp[mem\_size];

    for(int i=0;i<mem\_size;i++)

        temp[i] = mem[i];

    // copying into temporary array

    int tempp[p\_size];

    for(int i=0;i<p\_size;i++)

        tempp[i] = p[i];

    for(int i=0;i<p\_size;i++){

            int mx=0;

        for(int j=0;j<mem\_size;j++){

            if(temp[j] != 0 && temp[j] > temp[mx] && temp[j]>= tempp[i]){

                    mx=j;

            }

        }

        if(mx != 0){

            temp[mx] -= tempp[i];

            tempp[i] = mx;

        }

    }

    cout<<"Memory: \t";

    print(temp,mem\_size);

    cout<<"Processes: \t";

    print(tempp,p\_size);

}

int main(){

    cout<<"======>Segmentation-in-Memory<======"<<endl;

    cout<<"# Processes are allocated at the address/index of memory #"<<endl;

    int mem[] = {0,50,25,0,100,0,500,0,65,75,0,155,0},process[]={100,25,150,20,68};

    int lenm=13,lenp=5;

    cout<<"Memory: \t";

    print(mem,lenm);

    cout<<"Processes: \t";

    print(process,lenp);

    while(true){

        int ch;

        cout<<"1. First fit \t 2. Next fit \t 3. Best fit \t 4. Worst fit \t 5. Exit:";

        cin>>ch;

        switch(ch){

        case 1:First\_Fit(mem,process,lenm,lenp);

        break;

        case 2:Next\_Fit(mem,process,lenm,lenp);

        break;

        case 3:Best\_Fit(mem,process,lenm,lenp);

        break;

        case 4:Worst\_Fit(mem,process,lenm,lenp);

        break;

        case 5:return 0;

        break;

        default:cout<<"Enter 1/2/3/4/5"<<endl;

        break;

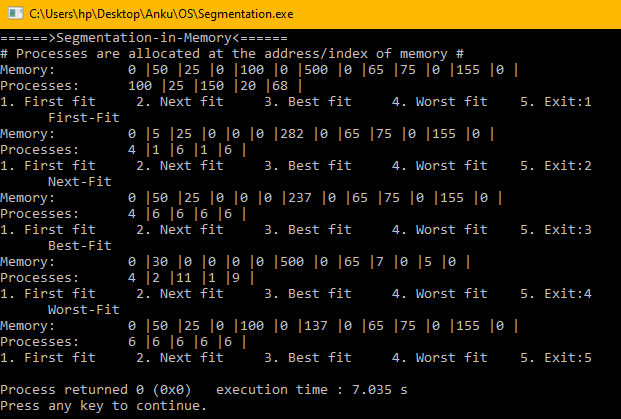
        }

    }

    return 0;

}

**Output:**

****

# **Lab 7**

**Q.Write a C program to simulate page replacement algorithms**

**(a) FIFO**

**(b) LRU**

**(c) LFU**

**Ans.** #include <iostream>

using namespace std;

bool check\_page(int page[],int n,int x){

    int f=0;

    for(int i=0;i<n;i++){

        if(page[i] == x){

            f = 1;break;

        }

    }

    if(f == 1)

        return true;

    return false;

}

void print(int a[],int s){

    cout<<"Pages: | ";

    for(int i=0;i<s;i++)

        cout<<a[i]<<" |";

    cout<<"  \*"<<endl;

}

void FIFO(int frms[],int no\_frames,int no\_pages){

    int j=0,n=no\_pages,m=no\_frames;

    float fault=0;

    int pages[n];

    //int A[m] = {0, 9, 0, 1, 8,1, 8, 7, 8, 7, 2, 1, 2, 3, 8, 2, 1, 7, 8, 2, 3, 8, 3,9};

    for(int i=0;i<n;i++)

        pages[i] = -1;

    cout<<"First-In-First-Out Page fault minimizing algorithm."<<endl;

    for(int i=0;i<m;i++){

        if(!check\_page(pages,n,frms[i])){

            pages[j] = frms[i];

            j = (j+1)%n;

            fault++;

            print(pages,n);

        }

        else

            cout<<"\t\tHit\n";

    }

    float fr = (fault/m)\*100;

    cout<<endl<<"Page Fault Ratio: \t"<<fr<<endl;

    cout<<"Page Hit Ratio: \t"<<100-fr<<endl;

}

int count\_min(int p[],int frm[],int i,int n,int m){

    int cnt[n][2];

    for(int i=0;i<n;i++){

        cnt[i][0] = p[i];

        cnt[i][1] = 0;

    }

    for(int k=i-1;k>=0;k--){

        if(check\_page(p,n,frm[k])){

            for(int j=0;j<n;j++)

                if(cnt[j][0] == frm[k])

                    cnt[j][1] += 1;

        }

    }

    int mn=0;

    for(int l=0;l<n;l++)

        if(cnt[mn][1] > cnt[l][1])

            mn = l;

    return cnt[mn][0];

}

int min\_count(int p[],int frm[],int i,int n,int m){

    int cnt[n][2];

    for(int i=0;i<n;i++){

        cnt[i][0] = p[i];

        cnt[i][1] = 0;

    }

    for(int k=i+1;k<m;k++){

        if(check\_page(p,n,frm[k])){

            for(int j=0;j<n;j++)

                if(cnt[j][0] == frm[k])

                    cnt[j][1] += 1;

        }

    }

    int mn=0;

    for(int l=0;l<n;l++)

        if(cnt[mn][1] > cnt[l][1])

            mn = l;

    return cnt[mn][0];

}

void LFU(int frms[],int no\_frames,int no\_pages){

    float fault=0;

    int n=no\_pages,m=no\_frames;

    int pages[n];

    int j=0;

    //int A[m] = {0, 9, 0, 1, 8,1, 8, 7, 8, 7, 2, 1, 2, 3, 8, 2, 1, 7, 8, 2, 3, 8, 3,9};

    for(int i=0;i<n;i++)

        pages[i] = -1;

    cout<<"Least-Frequently-Used(Optimal) Page fault minimizing algorithm."<<endl;

    cout<<"String : |";

    for(int i=0;i<m;i++)

        cout<<frms[i]<<" |";

    cout<<endl<<endl;

    for(int i=0;i<m;i++){

        if(!check\_page(pages,n,frms[i])){

            if(i <= n){

                pages[j] = frms[i];

                j++;

                fault++;

                print(pages,n);

            }

            else{

            // get the never used element

            int element = min\_count(pages,frms,i,n,m);

            for(int j=0;j<n;j++)

                if(pages[j] == element)

                    pages[j] = frms[i];

            fault +=1;

            print(pages,n);

        }

        }

        else{

            cout<<"\t\tHit\n";

        }

    }

    float fr = (fault/m)\*100;

    cout<<endl<<"Page Fault Ratio: \t"<<fr<<endl;

    cout<<"Page Hit Ratio: \t"<<100-fr<<endl;

}

void LRU(int frms[],int no\_frames,int no\_pages){

    float fault=0;

    int n=no\_pages,m=no\_frames;

    int pages[n];

    int j=0;

    //int A[m] = {0, 9, 0, 1, 8,1, 8, 7, 8, 7, 2, 1, 2, 3, 8, 2, 1, 7, 8, 2, 3, 8, 3,9};

    for(int i=0;i<n;i++)

        pages[i] = -1;

    cout<<"Least-Recently-Used Page fault minimizing algorithm."<<endl;

    cout<<"String : |";

    for(int i=0;i<m;i++)

        cout<<frms[i]<<" |";

    cout<<endl<<endl;

    for(int i=0;i<m;i++){

        if(!check\_page(pages,n,frms[i])){

            if(i <= n){

                pages[j] = frms[i];

                j++;

                fault++;

                print(pages,n);

            }

            else{

            // get the page which is not used for most upcoming requests

            int element = count\_min(pages,frms,i,n,m);

            for(int j=0;j<n;j++)

                if(pages[j] == element)

                    pages[j] = frms[i];

            fault += 1;

            print(pages,n);

        }

        }

        else{

            cout<<"\t\tHit\n";

        }

    }

    float fr = (fault/m)\*100;

    cout<<endl<<"Page Fault Ratio: \t"<<fr<<endl;

    cout<<"Page Hit Ratio: \t"<<100-fr<<endl;

}

int main(){

    int n,m;

    cout<<"Enter no. of page frames and no. of pages: "<<endl;

    cin>>n>>m;

    int A[m];

    cout<<"Enter the list of pages:"<<endl;

    for(int i=0;i<m;i++)

        cin>>A[i];

    while(1){

        int ch;

        cout<<"Enter 1.FIFO \t 2.LFU \t 3.LRU \t 4.Exit:";

        cin>>ch;

        switch(ch){

        case 1:FIFO(A,m,n);

        break;

        case 2:LFU(A,m,n);

        break;

        case 3:LRU(A,m,n);

        break;

        case 4:exit(0);

        break;

        default:cout<<"Enter 1/2/3/4 :"<<endl;

        break;

        }

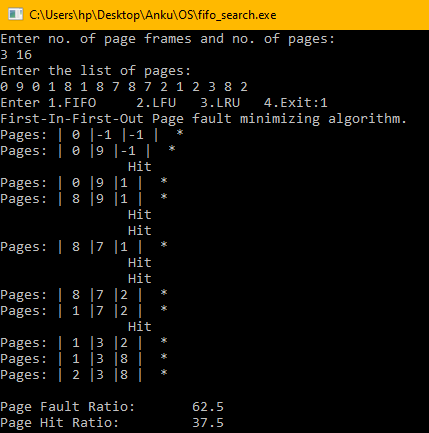
    }

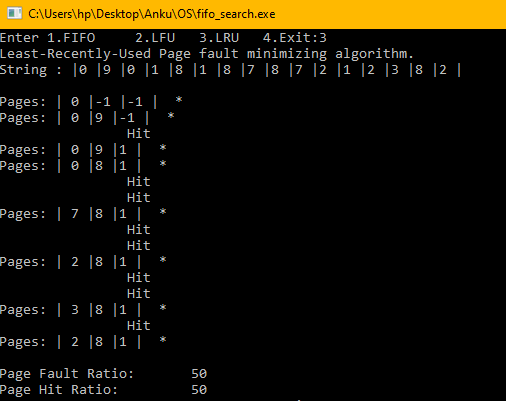
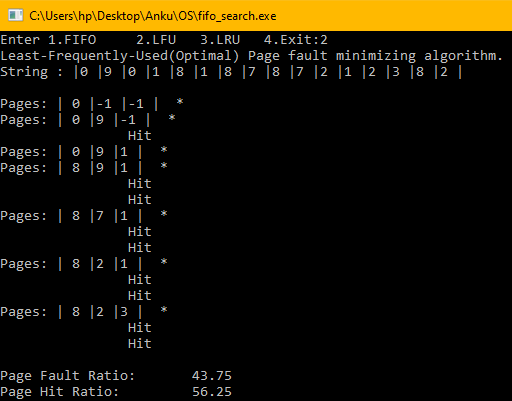
    //int A[] = {0, 9, 0, 1, 8,1, 8, 7, 8, 7, 2, 1, 2, 3, 8, 2, 1, 7, 8, 2, 3, 8, 3,9};

    return 0;

}

**Output:**

****

****