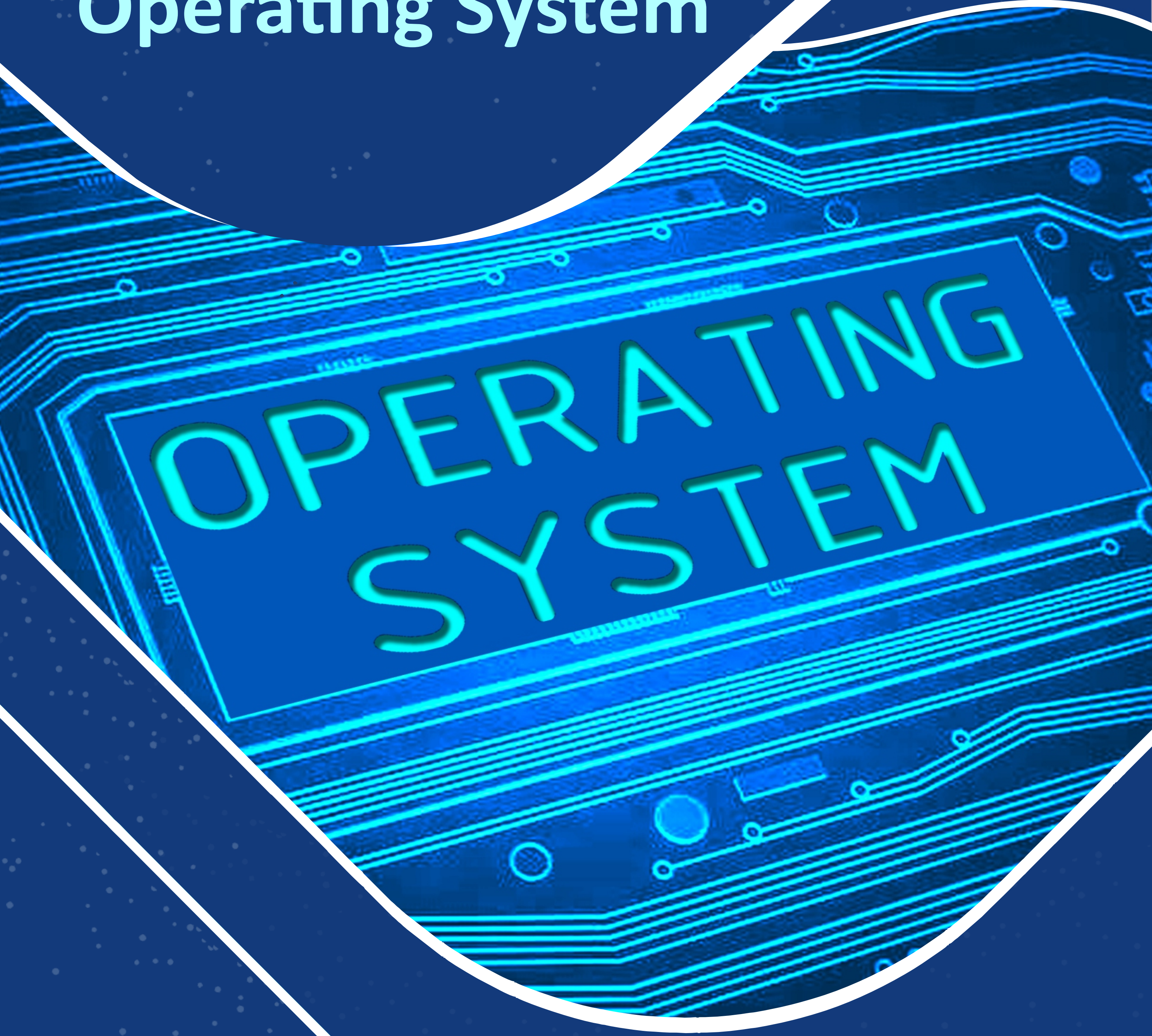
BAOU
Education for All

Dr. Babasaheb Ambedkar
Open University
(Established by Government of Gujarat)

# Fundamental of Operating System



MSCIT - 104

Master of Science
Information Technology

2021

# Fundamental of Operating System

Dr. Babasaheb Ambedkar Open University

# Fundamental of Operating System

**Course Writer**

| | |
|---|---|
| Dr. Neepa Shah | Associate Professor,<br>Dept. of Computer Science<br>Gujarat Vidyapith |
| Dr. Bhavik Pandya | Assistant Professor,<br>Navgujarat College of computer application,<br>Ahmedabad |
| Dr. Jitendra Bhatia | Assistant professor,<br>Computer Engineering Department,<br>Vishwakarma Government Engineering College,<br>Ahmedabad |

**Content Reviewer**

| | |
|---|---|
| Prof. (Dr.) Satyen Parikh | Executive Dean, Faculty of Computer Science and Application, Ganpat University, Kherva, Mehsana |

**Content Editor**

| | |
|---|---|
| Dr. Himanshu Patel | Assistant Professor,<br>School of Computer Science,<br>Dr. Babasaheb Ambedkar Open University |

Copyright © Dr. Babasaheb Ambedkar Open University – Ahmedabad. 2021

**ISBN -** 978- 81- 942146- 3-2

**Printed and published by:** Dr. Babasaheb Ambedkar Open University, Ahmedabad

While all efforts have been made by editors to check accuracy of the content, the representation of facts, principles, descriptions and methods are that of the respective module writers. Views expressed in the publication are that of the authors, and do not necessarily reflect the views of Dr. Babasaheb Ambedkar Open University. All products and services mentioned are owned by their respective copyrights holders, and mere presentation in the publication does not mean endorsement by Dr. Babasaheb Ambedkar Open University. Every effort has been made to acknowledge and attribute all sources of information used in preparation of this learning material. Readers are requested to kindly notify missing attribution, if any.

BAOU Education for All

Dr. Babasaheb Ambedkar Open University

MSCIT-104

# Fundamental of Operating System

## Block-1: Introduction of OS and Process Management

**UNIT-1**
Introduction of Operating System 02

**UNIT-2**
Process Concepts 15

**UNIT-3**
Threads 31

**UNIT-4**
Deadlock 52

## Block-2: Memory and File Management

**UNIT-1**
Memory Concepts 65

**UNIT-2**
Paging and Segmentation 81

**UNIT-3**
Virtual Memory 92

**UNIT-4**
File Concepts 105

## Block-3: Introduction to Linux/Unix

**UNIT-1**
Basics of Linux 122

**UNIT-2**
File System and Related Commands 135

**UNIT-3**
General Purpose Utilities Commands 154

**UNIT-4**
General Purpose Utilities on File 166

## Block-4: Process, File Management and Shell Programming in Linux/Unix

**UNIT-1**
Process Management and Related Commands 185

**UNIT-2**
Patterns Finding and Linux Filter Commands 201

**UNIT-3**
Inter-Process Communication 221

**UNIT-4**
Shell Programming 239

# Block-1

# Introduction of OS and Process Management

# Unit 1: Introduction of Operating System

1

## Unit Structure

1.1. Learning Objectives

1.2. Introduction

1.3. What is an Operating System?

1.4. A Processor

1.5. Interrupt Execution

1.6. Memory Hierarchy

1.7. Cache Memory

1.8. Input/Output Communication

1.9. GeneralCategories of Operating System

1.10. Let Us Sum Up

1.11. Glossary

1.12. References

## 1.1 LEARNING OBJECTIVE

- Understanding of the functions of the operating system
- Understanding of hardware and devices attached to the machine
- Memory hierarchy and its functions
- Different kinds of operating system

## 1.2 INTRODUCTION

Operating system is the one of the system software that appears when you start up your machine. This is the software which takes the control of the hardware and drive the system with user friendly way. There are two main entities of the system – server and clients. Requests are generated by the clients to the servers and result of the request pass to the client machine. While working on such tasks there are many parallel tasks are generated on the server machine and implicitly other processes that are working for the management of the server applications are working simultaneously in the environment. Management of the so many processes of the environment is the one of the task of the operating system. Another task is to manage memory and devices. Both type of memory are concern here primary and secondary memory. To create a file in the system, manage storage for the file and access of the devices in the system. Third important task of the operating system is to provide security and enhance treatment to the data. In this chapter we will study about primitive components of the operating system, concepts of the hardware, hierarchy of memory, device management and different kinds of operating system.

## 1.3 WHAT IS AN OPERATING SYSTEM?

We all are working with computers and at the initial task we have to work with the operating system, without operating system no one can move further with their applications. Hardware of the machine is inefficient to deal as a user interface. Many programs and data have also been entered to work together with use of hardware to fulfill a task; to manage hardware and different task we required a system called an operating system. There are many different task need to be handle by operating system, mainly – memory management, processor management, device management and data management. The figure 1.1 shows different parts of the operating system and its integration with other programs, hardware and users.

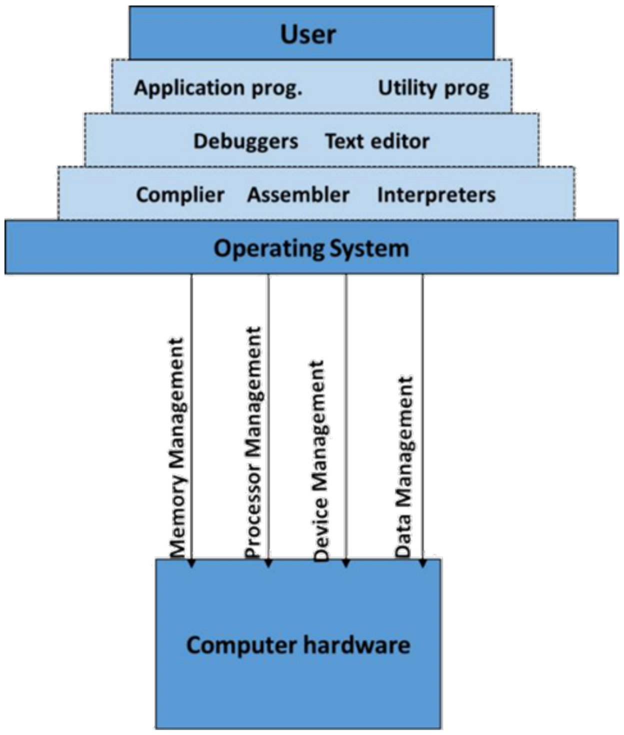


**Figure 1.1: Operating system layers and its functions**

Memory management. There are two type of memory one is primary memory and other is secondary memory. Primary memory is also known as Main memory deals with the data currently running processes. Unlike main memory, secondary memory is used as permanent storage of data. To create space for a file, storage is assigned on the secondary memory and to run a process, space is assign on a main memory. Both required proper management so files and processes get suitable treatment.

Processor management. A process created in the environment has life and during its life it need time to execute on the processor. Nowadays most of the systems are multi process system and many processes are running simultaneously in the operating system environment. At a time a processor can give a chance to a single process to execute on it. Execution of many processes is done by scheduling task. This scheduling task handle the time management for the processor.

Device management. Various types of input-output devices are working with the computer system. Every device need special methods for input and output data.

Some devices are elector magnetic while other are electro mechanical device, an operating system trap the signals generated by such devices and convey appropriate message to the user.

Data management. Files on the secondary storage and process on the primary storage need space to store its data. Both can expand and shrink in the environment so they need to have dynamic treatment for the space management of the data. File includes creation, expansion, shrink, remove, etc. operations and security of data, while process includes creation, control, inter-process communication, termination, etc. operations and protection private data.

Activity I
Find out what is POST (Power On Self Test) and what are the activities done during this

## 1.4 A PROCESSOR

A processor is a set of hardware including complex circuits including Arithmetic Logic and shift Unit (ALU). This unit is responsible to execute arithmetic instructions, logic instructions and shift instructions. These processor is now having special purpose instruction execution units like MFLOP (Mega Floating point instruction), IOP (Input-output processing), etc. there are many types of registers within the processor to execute these instructions. According to 8086 architecture given are basic registers and its functions.

| **Register Symbol** | **Name of the register** | **Number of bits** | **Details of the register** |
|---|---|---|---|
| IR | Instruction Register | 16 | To store the instruction before decode it. |
| AR | Address Register | 12 | To store the address of memory for operand<br>and location of instruction. |
| AC | Accumulator | 16 | Attached register with ALU to store the operand and/or store result. |
| DR | Data Register | 16 | To store the resultant data or second |

| | | | |
|---|---|---|---|
| | | | operand data of an instruction. |
| PC | Program Counter | 12 | To generate next instruction address. |
| TR | Temporary Register | 16 | To store intermediate data or address for<br>the execution of the instruction. |
| INPR | Input Register | 8 | To receive data from the input device. |
| OUTR | Output Register | 8 | To send data to output device |

**Table 1.1: List of Registers for the Basic Computer**

Activity II

a) Find the reason for numbers of the bits are different in different register of the processor.

b) How many registers are joined with the ALU?

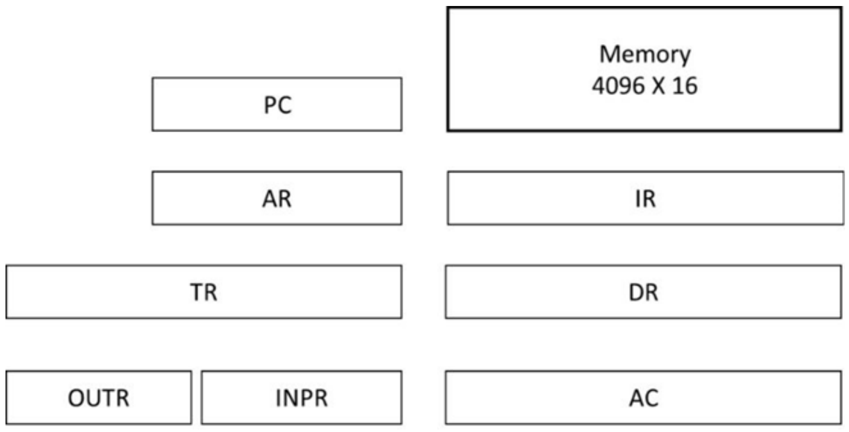


**Figure 1.2: Basic set of registers and memory**

Above figure 1.2 shows the basic register set and memory unit, these are vital components for execution of instruction in processor. Let us see how this instruction executes in using above components.

Execution of the Instruction. Program get space into main memory unit for the execution, so our instructions are in the memory and address of the instruction is stored in the Address Register (AR) which is given by PC to get new instruction address. New instruction is fetched from the memory by having an address of AR and stored in the IR. Here instruction get decoded by circuit and according to bits value operation code is generated for the execution of the instruction. This operation is known as micro-operation. Toexecute the micro-operation, processor uses a circuit (ALU) attached with AC. It stores operand in the AC and when more than one operand is required, it uses DR and other registers. Result generated after execution of the micro-operation is stored in AC. Given diagram of figure 1.3 shows the path of execution of instruction.

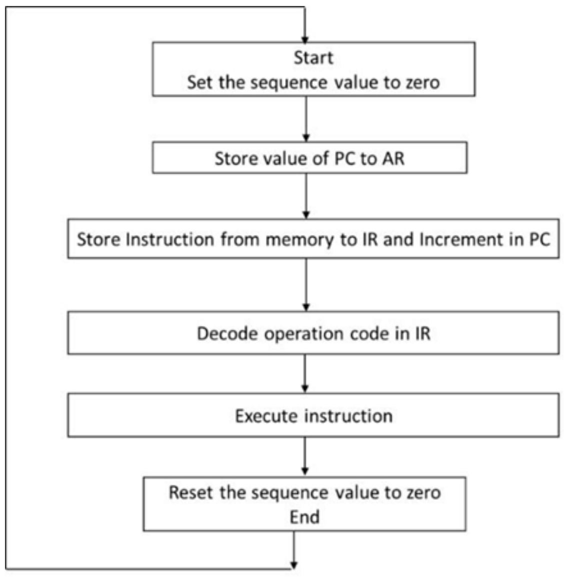

Start
Set the sequence value to zero
Store value of PC to AR
Store Instruction from memory to IR and Increment in PC
Decode operation code in IR
Execute instruction
Reset the sequence value to zero
End

**Figure 1.3: Steps of the Instruction execution using basic components**

## 1.5 INTERRUPT EXECUTION

Above figure just describe how instruction get executed in the environment. The computer checks flag bits to identify transfer of the program control. When program control of executing instruction is transferred to handle other utility known as interrupt. Computer cannot waste time to check flag bits and decides to transfer the control on each cycle of instruction execution given in the figure 1.3 an alternative to control such procedure is to let device or other services inform the computer when there is an interrupt in the environment. In the meantime computer can busy with its regular task. When interrupt is received by setting the flag, computer is informed for transfer of control. After completing the current cycle, new cycle begins to handle the interrupt. When interrupt is handled by processor it resume from the previous instruction where it has been left.

Figure 1.4 shows diagram for the interrupt execution, when interrupt has been identified after fetch and decode cycle, there are two task simultaneously going on in the processor- first is the completion of current instruction execution and second is change of the control from current instruction routine to interrupt execution routine.

Once interrupt routine address is given to the PC, next cycle will start from the first instruction of the interrupt routine.

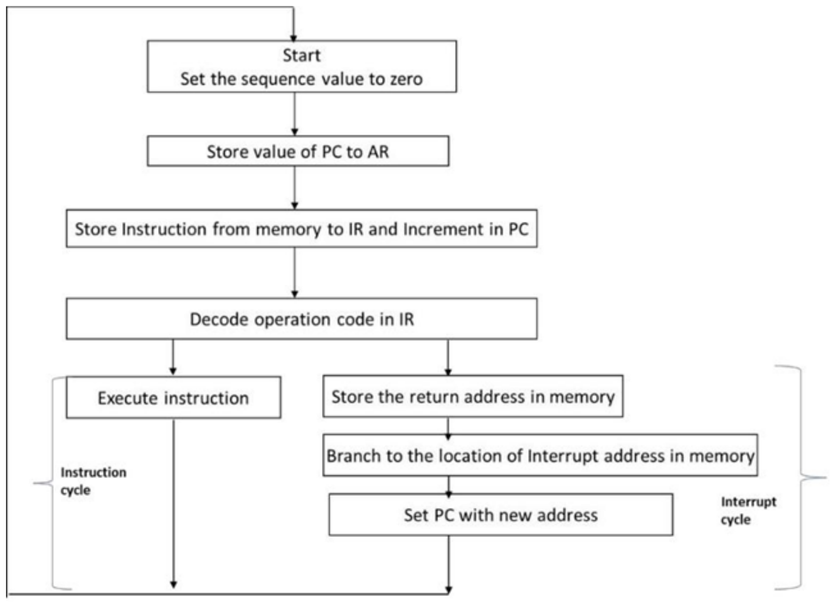

Start
Set the sequence value to zero
Store value of PC to AR
Store Instruction from memory to IR and Increment in PC
Decode operation code in IR
Execute instruction
Store the return address in memory
Branch to the location of Interrupt address in memory
Set PC with new address
Instruction cycle
Interrupt cycle

**Figure 1.4: Instruction execution and interrupt cycle Activity III**

Activity III

a) How the instruction cycle get executed in the environment?

b) What do you think where does the address of the branch stored in the computer? Why?

## 1.6 MEMORY HIERARCHY

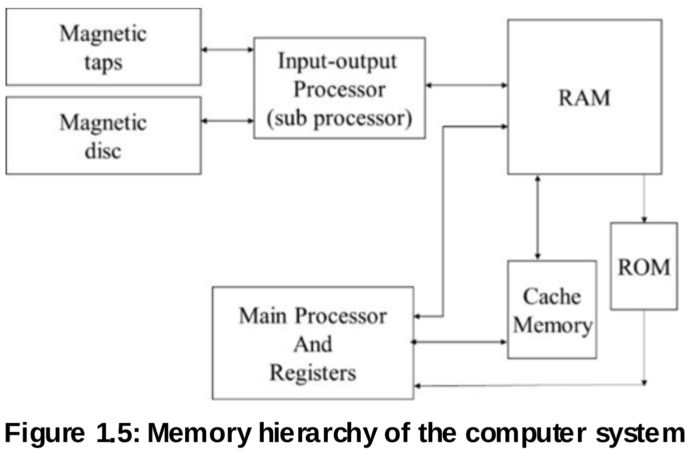

Magnetic taps
Magnetic disc
Input-output Processor (sub processor)
RAM
ROM
Cache Memory
Main Processor And Registers

**Figure 1.5: Memory hierarchy of the computer system**

Memory hierarchy is shown in figure 1.5. Registers inside the processor are the fastest storage components and cache memory is also works at the speed of the registers in the processor. Another memory component is RAM (Random Access Memory), this memory is slower than cache memory but it is more cost effective compare to cache memory. All the processors registers and devices are attached with this memory. Machine instructions are also stored in the RAM. It is known as main memory of the computer. ROM (Read Only Memory) is used as control memory where micro-instruction has been stored. This micro-instruction executes the micro-operations in the software based control unit (processor). Magnetic tap and magnetic disc are permanent storage of the data. This memory is very slow speed components but it preserve the data even after power supply has been stopped. All the data is stored in this memory as a file. This memory is known as secondary memory or auxiliary memory. It has large capacity for storage and cost effective compare to other memory components.

## 1.7 CACHE MEMORY

Cache memory is found between processor and main memory. Generally speed of the cache memory is many time higher than a main memory. Size of the cache memory is small that is why part of the program is stored within the cache memory. Processor has direct access of the cache memory and speed of the cache memory is as fast as processors registers so execution of the instruction can be done at the fastest rate when cache memory size is good enough.

When processor needs to access memory, first it examined the cache memory. If it is not found in the cache memory than main memory is accessed for that location. That memory word or instruction is then transferred from main memory to the cache memory. When a memory word is found in the cache memory, it is called hit and if it is not found in the cache memory than it is called miss. The performance of the cache memory is measured in terms of the hit ration.

The formula for the hit ratio is given below:

$$hit\ ratio = \frac{numbers\ of\ hit}{numbers\ of\ hit + numbers\ of\ miss}$$

Normally hit ratio higher than 0.9 is considered as a good rate for any processor. Higher the value higher the locality of reference. Locality of reference is references to memory at any given time interval tend to be confined within a few localized area of memory. The purpose of having cache memory is fast access time. To achieve this very little time or no time must be wasted when searching for the words in the cache memory. The process to search a word in the memory is called mapping process.

Activity IV

a) Put the memory component in ascending order of its speed.

b) Why main memory and cache memory both are connected to the processor?

c) What is miss? If it is miss how it is resolved by processor?

## 1.8 INPUT/OUTPUT COMMUNICATION

When processor is attached with various devices, it requires a method to communicate with the devices. An interface between processor and input-output device has been developed so by using this interface data can transferred and received from both end. This interface provides methods for transferring information between internal storage and external storage. Following are the reasons to develop an interface between interface between internal storage and external storage to resolve the differences:

- External devices are electromagnetic and electromechanical while processor is electronics device. So conversion of signal values required.
- Data transfer rates are different so synchronization is required.
- Data codes and formats of external devices are different then processor word format so conversion is needed.

The interface can be done by I/O bus consists of data line, address line and control line shown in the figure 1.6. Each peripheral device has associated with it an interface unit. It synchronize the data flow and supervises the transfer between external device and internal device. When the address is available in address Line, the processor provides a function code on the control line. This function code is referred as an I/O command.

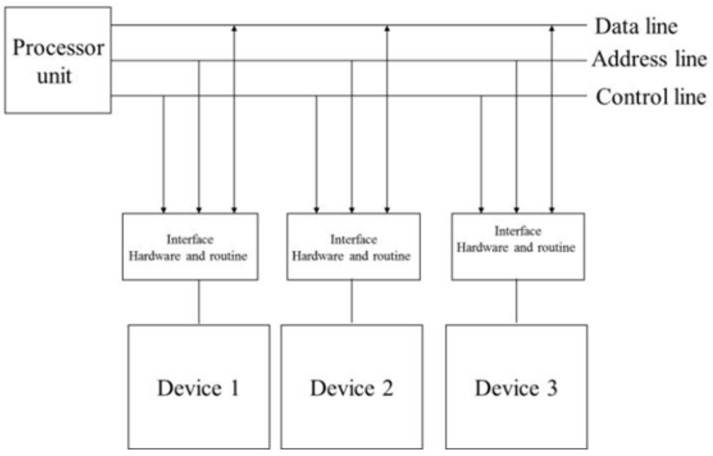

Processor unit
Data line
Address line
Control line
Interface Hardware and routine
Interface Hardware and routine
Interface Hardware and routine
Device 1
Device 2
Device 3

**Figure 1.6: Interface between internal and external devices**

Activity V

a)Define the functions of interface unit.

## 1.9 GENERAL CATEGORIES OF OPERATING SYSTEM

In above sections, we have introduced the general structure of a typical computer system. We have also seen that how the peripheral devices are attached to the processor and I/O between external and internal storage. There are different categories of the operating system according to the use and utilities provided by architecture.

**Desktop system**. The program control the machine itself and provide services to the user of the machine only is called desktop or laptop operating system. This operating system takes the control of the hardware and run the environment to provide services like – memory management, process management, device access and data handling. Many systems of this category provide security and data protection. Examples are windows 10, Ubuntu 18.04 LTS, Mac OS 10.14 (Mojave) etc.

**Multiprocessor system**. System with two or more than two processors is known as multiprocessor system. This system shares common bus, clock, memory and devices. According to Flynn’s classification, MISD (Multiple Instruction stream and Single Data stream) and MIMD (Multiple Instruction stream and Multiple Data stream) computers are of this category. Distributed system and clustered system are also part of multiprocessor system. When data stream and instruction stream are increased it effect the following parameters:

1. **Throughput**. Get more work done in less time. When more data and instruction stream are there they can do the work simultaneously so the performance is increased. The speed-up ratio is increased for N processor work together but theoretically we are not getting increase according to number of processors. There are many reasons for that.

2. **Economy scale**. MISD and MIMD architecture can use cluster of the computers and processors, that sharing external devices. This can be a cost effective solution in many cases.

3. **Reliability**. Failure of one processor will not affect other processing going on in the system, albeit load of the failed processor can be divided among other processors. This is user transparent mechanism so user don't have to bother about failure and system will be more reliable.

**Distributed system**. Distributed system is collection of physically separate, heterogeneous, and system that are networked to provide services. Sharable devices increase the speed of computation, data availability and reliability.

**Clustered system**. Cluster is usually used to provide high-availability services, when one of the node in the cluster is failed than other node take charge of that node and start execution from where it is failed. There are mainly two types of clustered system- asymmetrical and symmetrical. In asymmetrical one node is standby mode so it can take the control of the failed node while symmetrical cluster all nodes are running simultaneously and monitor each other.

**Real time system**. Real time system are special system that works on special requirements. The unique feature of this system is to complete the task in given deadline and instant service to the requisition generated in the environment. If task is not finished or handled in the given timeframe than it can cause the disaster. Major issue in the real time system is to develop a routine for proper scheduling of processes.

**Handheld system**. This category of the system includes PDA (Personal Digital Assistant) like tablets, cellular telephones etc. main issue of this system to manage data and application in limited amount of resources. Size of the device is small, processor rate is also slow, and amount of the memory is small to manage data.

Activity VI

a)Get information about all types of operating system and give examples of each one.

## 1.10LET US SUM UP

- Functions of the operating system are memory management, process management, device management and data management
- Processor and basic set of registers with its function.
- Instruction execution cycle and handling of the interrupt
- Different kind of the memory and its usage – main memory, auxiliary memory and cache memory
- Usage of cache memory and hit ratio
- Input output communication using an interface that include data line, address line and control line.
- Different category of operating systems and its applicability.

## 1.11 GLOSSARY

| | |
|---|---|
| IR | Instruction Register |
| AR | Address Register |
| AC | Accumulator |
| DR | Data Register |
| PC | Program Counter |

| | |
|---|---|
| TR | Temporary Register |
| INPR | Input Register |
| OUTR | Output Register |
| ALU | Arithmetic Logic Unit |
| MISD | Multiple Instruction stream and Single Data stream |
| MIMD | Multiple Instruction stream and Multiple Data stream |
| PDA | Personal Digital Assistant |

## 1.12REFERENCES

(1) Operating Systems Concepts. Addision – Wesley By Silberschetz A and Galvin.

(2) Operating Systems Prentice Hall of India Pvt. Ltd. By Tanenbaum.

(3) Computer Architecture – Pearson Education by Morris Mano.

2

# Unit 2: Process Concepts

## Unit Structure

2.1. Learning Objectives

2.2. Introduction

2.3. The Process

2.4. State of the Process

2.5. Process Scheduling

2.6. Context of the Process

2.7. Process Scheduling Criteria

2.8. Scheduling Algorithms

2.9. Let Us Sum Up

2.10. Glossary

2.11. References

## 2.1 LEARNING OBJECTIVE

- Different states of the process
- Creation of the process and allocating resource to the process
- Scheduling of the process
- Algorithm for scheduling process

## 2.2 INTRODUCTION

Multiuser system and multitasking system are having more than one process in the environment for the execution. Processor can take only one process at a time for the execution. When one process is during I/O task at that time processor is ideal and take another process in the environment for the execution.

This way simultaneous execution is possible. One of the important task for the operating system is to execute a process and give resources for the execution of the process. When process is created in the environment all resources required to execute a process is given to the process and when it is terminated resources should be revoked from the process. This chapter will give an idea about process creation, execution task of the process and scheduling of a process when many processes are working simultaneously in the system.

## 2.3 THE PROCESS

Process is execution of program on processor using main memory and set of registers. This resources are provided by operating system when it get created in the environment. When process get birth on the process it give resource in term of storage space on main memory, secondary memory, registers and many operating system data structures get updated for the new born process. When it is created in the environment operating system gives space in four different way. First it gives text region to store the data of program instructions. Instructions of Executable file is stored within this region. Another is data region, where variable of the process get space to store data. Stack region is contains address of calling function and parameters passes across the functions. A process may have heap section in which memory that is dynamically allocated during the execution of the program is given space. The structure of the all regions is shown in figure 2.1.

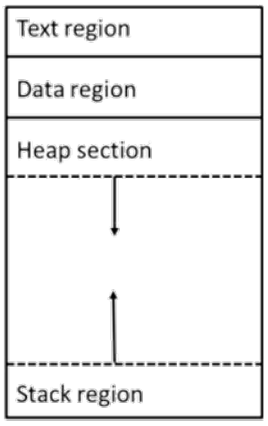


**Figure 2.1: Different region of the process in memory**

## 2.4 STATE OF THE PROCESS

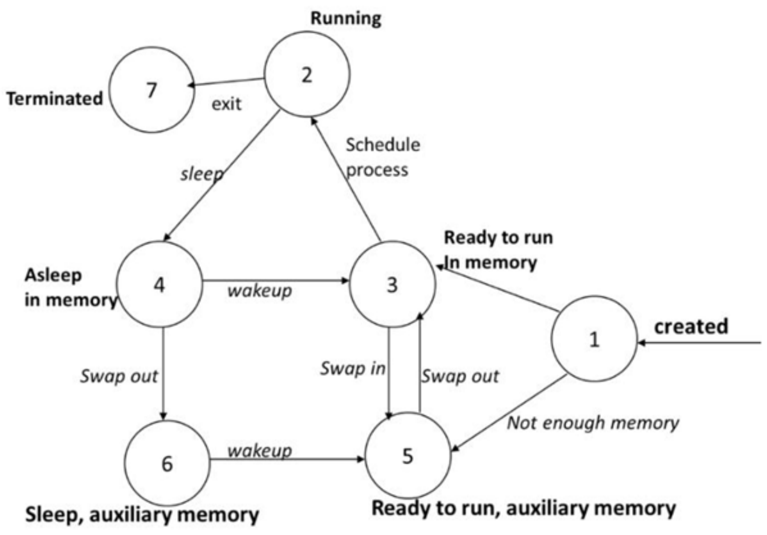
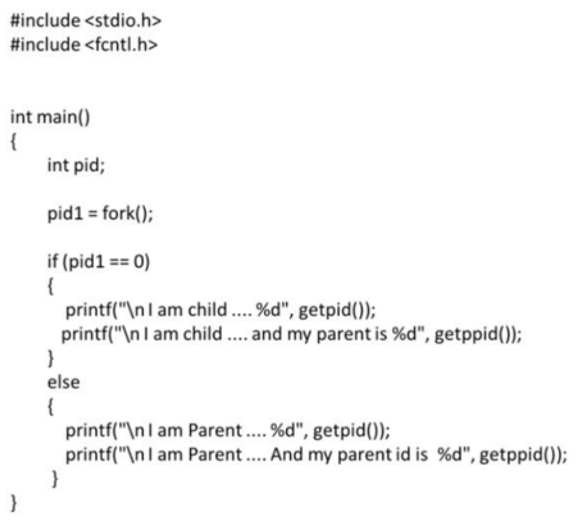


**Figure 2.2: Diagram of the process state**

When process get birth in the environment, it has different state during execution. When process get birth it is called created in the environment, this is the first state of the process. If there is enough memory for process to store its region it will get space in the main memory otherwise process get created but accommodate on the auxiliary memory. This memory is known as swap memory in UNIX operating system while windows operating system called it virtual memory. When process in memory and not executing than the state is called reedy to run state. Whenever process become most eligible for execution the state will be changed and it became running process in the environment. Sometimes process may not get resources which is accessed by other process in the environment so process may have to wait for the resource become free, it is the time process is in waiting state or sleeping state. A waiting process/ sleeping process sometime transferred to the auxiliary device due to lack of memory. Whenever operating system get room for the process, it calls the process in the main memory so sleeping process state changing from sleeping in main memory to sleeping in auxiliary memory.

Let us take an example of process on UNIX operating system when get created by using fork () command. Fork command creates new process and new process will be a child process of currently running process.

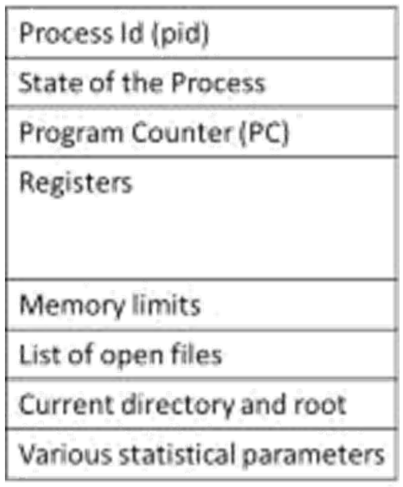
Given program is creating a child process in the environment. UNIX operating system uses fork () to create a new process, in this program fork () command return a process id of new created process. Newly created process is known as child process and after this command there are two process in the environment. Processor may select any one on random bases to the execution. While running this program in the UNIX environment, either you get first control of child or parent. In the parent process value of the variable pid is some positive number because parent has created a child process and operating system assign a id of this process, while in child process value of variable pid will be zero. Another functions getpid () and getppid () give process id of current running process and parent of current running process respectively. While running this program you will get output different state of the process. When processor is executing one process at that time other process is waiting/sleeping state. Before getting a turn for execution of a process, it come to the state ready to run than process will get selection for execution by scheduling algorithm.



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


int main()
{
      int pid;

      pid1 = fork();

      if (pid1 == 0)
      {
         printf("\n I am child .... %d", getpid());
        printf("\n I am child .... and my parent is %d", getppid());
      }
      else
      {
         printf("\n I am Parent .... %d", getpid());
         printf("\n I am Parent .... And my parent id is  %d", getppid());
       }
}
```

**Figure2.3: A program of creation of a child process using fork ()**

Above program will give you most of the state of process when it is executed by a user. Other than time for execution and different state, a process also need space for various computing purposes. It is known as Process Control Block (PCB) of also as a task control block. Figure 2.4 shows content of the PCB.



| |
|---|
| Process Id (pid) |
| State of the Process |
| Program Counter (PC) |
| Registers |
| Memory limits |
| List of open files |
| Current directory and root |
| Various statistical parameters |

**2.4: Process Control Block (PCB)**

Process id: every process is given a unique id in the environment of operating system. Management of the process is done by using this id. It is also called process number.

Process state: PCB stores state of the while process is switching from one state to another during the execution. It is important to know the current state of the process for executing other process in the environment and inter process communication.

Registers of the processor. Number of registers in the process can be vary as its depend on the type of the processor but information about all general purpose registers come under the context of the process so when there is a context of the process is changed, all information regarding running process including registers of the processor's information is stored and used when it resume in the environment.

Scheduling parameters. Operating system schedule each process in the round robin fashion. It depend on parameters like age of the process, priority of the execution of the process, etc. scheduler choose the process on the base of such parameters and apply the algorithm.

Memory management information. While using primary memory for the execution of the process in the running state, it also may swap out to the auxiliary memory when it is waiting or sleeping. Operating system must know the memory occupied by the process and where it is located while execution of the process.

Accounting information. Accounting information contains CPU usage of the process, limits of the process in the environment for the execution, real time used and actual used time, and many more parameters about maintaining a statistic of a process.

Information about I/O. Information about I/O done using various device during the execution of the process.

Activity I

a) Discuss different number of states during execution of a process?

b) Is it possible to schedule a process from the state sleep in virtual device (auxiliary memory) to running state? Why?

c) Write a program to find maximum number of processes created by a user in the environment.

## 2.5 PROCESS SCHEDULING

Kernel runs in the environment as a collection of system process. One of those process is schedule which decide the turn of the next coming process for a processor. The selection of appropriate process is carried out by a scheduler on the base of various parameters. There are two types of the scheduler – long term processor and short term processor. Long term scheduler schedule processes from the pool of processes created by mass-storage device (magnetic disk) and place them into memory. It is also called job scheduler. Short term scheduler schedule process from the ready to run state to running state, also known as CPU scheduler.

There are many criteria to switch over from one process to another. Sometime process is waiting for the resources become free or time slot given by operating system to run in the environment is over or higher priority process has come into the environment. All this circumstances is can cause the process to stop its execution and give turn to the other process to execute. Later one the process will resume from the current state only.

## 2.6 CONTEXT OF THE PROCESS

When process has been created in the environment, it consume memory in different regions like text, data and stack and heap. Executing process has information related to user level that is about user id, I/O parameters, and environment variables etc. this all make context of the process for user level. Similarly when operating system running the process in kernel level while execution of the system calls, it generates context of the process for kernel level. Execution of the process uses different registers of the processor so another context of the process is register level context.

**Context switch**. As we have discussed earlier that process switch from one process to another. While moving from the one process to another, it stores the context of all levels in memory and established a context of next coming process in the environment and resume the execution of that process. It is called context switch.

## 2.7 PROCESS SCHEDULING CRITERIA

Many criteria have been suggested by different scheduling algorithms for comparing more process in the environment. The criteria include following things:

**CPU utilization**. To keep CPU as busy as possible, we consider utilization of the CPU. It can be range from 0 to 100 percent.

**Throughput**. If CPU is executing processes one by one then work is done by processor. Number of processes is completed per given time unit is called the throughput.

**Turnaround time**. It is always important to know how long a processor takes to execute a process. Time span taken by a process from get created in the environment of terminates is considered as turnaround time.

**Waiting time**. Amount of time spend for running state of the process and I/O of the process never been affected by CPU-scheduling algorithm but time spend for waiting in a reedy run state of a process affect the process. Waiting time is sum of such period in a ready to run state.

**Response time**. Many times a process produce output early and further continue to compute next result. This measure is the time required to production of result from first response to the last response, is called the response time.

It is desirable to maximize CPU utilization and throughput time and minimize turnaround, waiting and response time. Under certain circumstances, it is desirable that rather than maximize or minimize; it apply to average measure. However it is always better to optimize by maximize and minimize measure.

Activity II

a) What is context switch and list out event when process does a context switch?

b) Differentiate CPU time and Turnaround time

## 2.8 SCHEDULING ALGORITHMS

### 1. First-Come, First-Served Scheduling

First-Come, First-Served (FCFS) is the simplest algorithm in which the process generated first in the environment served first by the processor. CPU schedule the first process from the queue. This queue is managed by scheduler and new process inserted at the rear while processor serve the process from the front. The PCB of new coming process attached at the tail of the queue. When CPU become free it will take another process from the queue. The negative side of the algorithms, its average waiting time is quite long. Let's understand using one example.

| Proces | Burst time (milliseconds) |
|---|---|
| p1 | 22 |
| p2 | 05 |
| p3 | 05 |
| p4 | 04 |

Process – Process end time: p1 – 0, p2 – 22, p3 – 27, p4 – 31.
Thus, for above example the average waiting time is (0 + 22 + 27 + 31)/4 = 20 milliseconds.

Suppose, the order of the process is altered like – p2, p3, p4, p1 then average waiting time will be (0 + 5 + 10 + 14) = 7.25 milliseconds.

## 2. Shortest-Job-First Scheduling

This algorithm follows different rule for scheduling process for the CPU. To overcome of the problem of FCFS algorithm, this Shortest-Job-First (SJF) uses time parameter for process to schedule for the CPU. It takes shortest process first for the execution and longest process at last. When two or more processes has similar time span than it apply FCFS algorithm to break the tie.
To understand the function of this algorithm, let's discuss one example.

| Proces | Burst time (millisecond) |
|---|---|
| p1 | 12 |
| p2 | 15 |

|  |  |
|---|---|
| p3 | 03 |
| p4 | 02 |

According to time taken by a process sorted list of the given process would be as under:

Process – process time: p4 – 02, p3 – 03, p1 – 12, p2 – 15

Thus, for above example the average waiting time is (5 + 17 + 2 + 0)/4 = 6 milliseconds. The waiting time for the p1 is 5 milliseconds, waiting time for p2 is 17 milliseconds, 2 milliseconds is waiting time for p3 process and 0 is the waiting time for the p4 process.

It is proven that SJF algorithm is optimal, it gives minimal average waiting time. Putting shortest process near the front end and after that long process decreases the waiting time of the short processes more than increases the waiting time of long processes. Thus, average waiting time is decreasing. Although the SJF algorithm is optimal but it cannot be implemented at short-term CPU scheduling level.Once the process given to the short term scheduling there is no way to know the length of the next CPU burst. In short term scheduling, we don’t know the length of the next CPU burst. We can estimate that by predicting that based on previous ones.

SJF algorithm can be either preemptive or non-preemptive mode. When new process arrives, it is in ready to run state while some previous process is still executing in the environment. On the next burst of CPU, if newly arrived process is shorter than currently running process than SJF will take newly arrived process from the queue if the mode of the algorithm is preemptive. In non-preemptive mode SJF algorithm will allow the currently running process to finish its work with the CPU. Preemptive SJF scheduling is sometime called as shortest-remaining-time-first scheduling. Let’s understand with an example.

| Process | Arrival time (24illisecond ) | Burst time (milliseconds) |
|---|---|---|

| | | |
|---|---|---|
| p1 | 0 | 08 |
| p2 | 1 | 15 |
| p3 | 2 | 03 |
| p4 | 3 | 09 |

According to time taken by a process and arrival time of the process given process would be as under:

Process – process start time: p1 – 0, p3 – 02, p1 – 05, p4 – 11, p2 – 20. So the average waiting time for this example is 20/4 = 5 milliseconds.

## 3. Priority Scheduling

Priority scheduling algorithm is associated with the priority of the process, scheduler will schedule the process with the highest priority process first. Again equal priority process will be schedule on the base of the FCFS algorithm. In this algorithm priority is given to each process by some fixed numbers. It is up to the system that how it treats the priority of the process, some operating system take lower value of priority has highest chance to get schedule first while some has higher the define value of priority make the process schedule first. Given table is an example of the process and its priority.

| Process | Burst time (milliseconds) | Priority |
|---|---|---|
| P1 | 5 | 4 |
| P2 | 2 | 1 |
| P3 | 5 | 5 |
| P4 | 1 | 3 |

According to algorithm we would schedule the processes like p2 – 0, p4 – 5, p1 – 6, p3 – 11. Priority of the process is determined on the base of various parameters like time limits, memory requirements, the number of files opened, ratio of average I/O burst to average CPU burst etc. this algorithm can be either preemptive or non-preemptive. When new process arrives at the queue in ready to run state, its priority is compared with the currently running process. If newly arrived process priority is higher than currently running process than scheduler preempted the running process and schedule new process. Non-preemptive algorithm will put the newly higher priority process at the front of the queue.

A major problem of the algorithm is indefinite blocking or starvation. A process with lower priority value is in ready to run state but scheduler won’t schedule it because of other higher priority processes are already there and process is in waiting state for indefinitely. A solution of this problem is provided by aging parameter. Aging is a method that gradually increases priority of the process that had been in waiting state for long back. We can put the aging bar so no process will have age more than this specific value and thus process will get the chance first due to is aging parameter having large value.

## 4. Round-Robin Scheduling

Round-Robin scheduling algorithm is specially designed for time sharing systems. This algorithm is very similar to the FCFS algorithm. Here preemptive execution of a process is allowed so processes are switching during the execution in the system. Time quantum is assign to each process for staying in a running mode, after this quantum one process scheduler should do context switch so next process will come for the execution from the queue. This queue is maintain in a FIFO manner, so after the context switch, dispatched process will add at the tail of the queue and next process will take in for the CPU running state from the head. After every time quantum such context switch occur in the environment. This time quantum is also known as time slice.

The average waiting time for this scheduling algorithm is often long. Consider given example to understand the scheduling process.

| Process | Burst time |
|---|---|
| | (milliseconds) |
| p1 | 12 |

| | |
|---|---|
| p2 | 15 |
| p3 | 03 |
| p4 | 02 |

For given example if time quantum is 3 milliseconds than each process will get 3 milliseconds to stay in running mode after that there will be a context switch. So first process will get first chance because it is the first process of the FIFO stack. After 3 milliseconds next process will be taken from the head and current process will be attached to the tail of queue. Here is the schedule for processes of round-Robin algorithm.

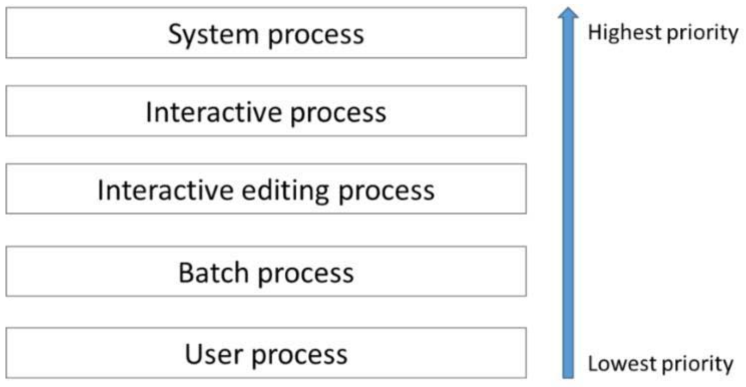
Process – Process schedule time: p1 – 0,p2 – 3, p3 – 6, p4 – 09, p1 – 11,p2 – 14, p1 – 17, p2 – 20, p1 – 23, p2 – 26,p2 – 29.

So waiting time for process p1 is 14 that is (0 + 8 + 3 + 3), process p2 is (3 + 10 + 3 + 3) = 19. Waiting time for process p3 is 6 and p4 is 9. Thu average waiting time is (14 + 19 + 6 + 9) = 48/4 = 12 milliseconds.

The performance of the Round-Robin algorithm depends on the size of the time quantum. If time quantum is large than function of the algorithm is as good as FCFS. If it is small than this techniques is called process sharing; if n process in the queue, 1/n the speed of the real processor.

## 5. Multi queue scheduling

This scheduling algorithm works when processes can easily classified into different groups. Suppose foreground processes means the process is interactive and background process means the process works in a batch. Such processes have different response-time requirements and also different scheduling needs. Foreground process have higher priority over background process. This algorithm partition the ready to run processes into several queues based on memory size, process priority, process type.



2.5: Process Priority and Its Execution Order

As shown in the example user processes have lowest priority. Batch processes have second highest priority. Interactive editing and interactive processes come higher than batch processes. System processes have the highest priority. When higher priority processes are executing in the environment than lower priority processes are preempted in the environment.

## 6. Multilevel Feedback Queue Scheduling

In multilevel queue scheduling algorithm processes are permanently assigned to a queue when it is created in the environment. There are separate queue for presses according to their functions. Suppose one process cannot move from one queue to other because it doesn’t change its nature of function. This would take us to the advantage of low scheduling overhead but also provide inflexibility.

Unlike this, multilevel feedback queue scheduling algorithm allows process to move between queues. Migrate a process from one queue to another queue is based on the CPU burst time. If a process is taking much time in running state than it will be migrated to the low level priority queue. This makes the I/O bound and interactive process in the higher priority. Apart from migrating process to low level priority it also move process that is in ready to run state since long time, in the higher priority queue. This way starvation due to aging can be prevented. Let’s understand with an example; suppose there are three queues for the process to get schedule. The scheduler will execute processes from the very first queue. When this queue become empty it will take next process from the next queue. In between, a process with higher priority come to the above queue than this process will be preempted scheduler will schedule newly coming process.

A process created in the ready to run mode is put in the first queue. Suppose time quantum is given for 4 seconds and if the process won’t able to finish within the time slice than this process move to the next queue and put at the tail of the queue. Here time quantum of the process execution will be increased. If it is not successes in the next queue to complete its execution than it will be again moved to another queue after this. Thus, if a process having CPU burst time more than 4 milliseconds than it has highest chances to get executed first in the queue. Long processes automatically served as FCFS order.

Generally multilevel feedback queue scheduler is defined by given parameters:

- Number of queue in the environment.
- Scheduling algorithm for each queue
- Method to migrate a process to higher priority queue and same to migrate to lower priority queue
- Method to determine correct queue for process when it required services

Activity III

a) If process P1 need 24 ms, P2 needs 3 ms and P3 needs 3 ms for the execution in the FCFS scheduling algorithm then find out average waiting time of the system.

b) Given is list of process and its execution time for SJF scheduling algorithm. P1 – 6ms, P2 – 8ms, P3 – 7ms, P4 – 3ms. Find average waiting time for the system

c) Find out average waiting time for Round Robin scheduling algorithm for the data given in activity III. (a).

## 2.9LET US SUM UP

- Different state of the process life cycle
- Text region, data region, stack region and heap area of the process.
- Context of the process and context switch in the environment
- Function of the scheduler and scheduling parameters
- Scheduling algorithms like first-come first-served, shortest job first, priority scheduling, round-robin scheduling, multilevel queue scheduling and multilevel feedback queue scheduling.

## 2.10 GLOSSARY

| | |
|---|---|
| PCB | Process Control Block |
| FCFS | First Come First Serve |
| SJF | Shortest Job Fist |

|  |  |
|---|---|
| RR | Round Robin |

## 2.11REFERENCES

1) Operating Systems Concepts. – Addision – Wesley by Silberschetz A and Galvin.
2) Operating Systems-Design and Implementation – Prentice Hall of India Pvt. Ltd. By Tanenbaum and Albert S. Woodhull.
3) Computer Architecture – Pearson Education by Morris Mano
4) Modern Operating Systems – PHI By Andrew S. Tanenbaum

# Unit 3: Threads

3

## Unit Structure

3.1. Learning Objectives

3.2. Introduction

3.3. Introductionof Threads

3.4. Benefits of the Thread

3.5. Issues of Multicore Programming

3.6. Multithreaded Models

3.7. Program of Threads

3.8. Let Us Sum Up

3.9. Glossary

3.10. References

## 3.1 LEARNING OBJECTIVE

- Introduction of the thread and processor utilization
- Types of threads
- Different type of model using thread
- Program of threads

## 3.2 INTRODUCTION

Management of process is usually considered with classical operating system. When input-output processing is going on in the environment than processor become idle. We can utilize this time of processor by giving some task during time. Implementation of this idea is possible when we can divide a whole process into small tasks and when one task is working with one processing element simultaneously other task can be handled by second processing element in the environment. Many distributed operating system working with similar environment. This ability provides many advantages but same time it also having its own problems. In this chapter we will study these issues and methods to organize processor and processes. We will also study about scheduling environment of multiple processing.

## 3.3 THE PROCESS

Classical operating systems has single thread of control. Actually there are many situations where it is desirable to have multiple threads of single process can execute simultaneously. When any thread of a process is sleeping or doing some I/O task and processor is free then another thread of the same process can got processor time. Thus the net result of the process is improve by higher throughput. This is beneficial because here both threads are using common buffer cache and two simultaneous execution is possible. When we have parallel server environment than it is difficult to execute tasks of same process in a two different server due to data sharing. Here same buffer cache will support data sharing.

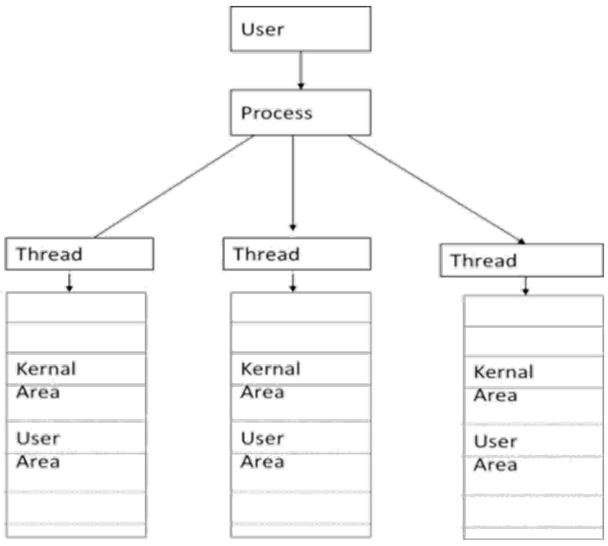


**Figure 3.1: Hierarchy of User to Thread in Operating System**

Figure 3.1 shows a process with three threads running on the machine. Threads are also known as lightweight processes. Threads also run in sequential manner like a process in the environment. It has own program counter and stack to keep track of where it is. Each thread has kernel area and user area for the execution of instructions. Like process, threads also share processor. It can also create child process and can block waiting for system calls to complete. When one thread is sleep mode another thread of the same process can execute in running state.

All threads have common address space, which means they share the same user and kernel environment. There is no protection between threads is not possible to maintain data sharing when one process is executing in a different task simultaneously. In addition to address space, threads share open files, child process and signals. So when threads are running in the environment, they are closely executing instructions with support of each other. Thread can also have states like – ready to run, sleeping, terminated or running.

## 3.4BENEFITS OF THE THREAD

We can divide the benefits of the multithreaded process into four categories – Responsiveness, sharing of resources, economy effects and scalability of architecture. Let’s discuss each with details.

Responsiveness. Multithreaded process allow to run a part of process in the environment even one part of the process is blocked or performing the lengthy task. This is the reason that response of the process is increased for the user. For example when a thread of an editor process is busy with find and replace utility, other thread can print the data for the same process.

Sharing of Resources. Processes may share resources with different techniques like shared memory, message passing etc. This techniques are developed by programmer but as discussed above thread is sharing common address space and resources. This will increase the throughput of the process and each thread can use the data from common address space.

Economy. Allocation of resources and address space is costly operation in the environment for a process. It is time consuming process to allocate and deallocate resources to a process. When we divide

a process into thread, it can share resources given to a process among the different threads. This way we can reduce the overhead of allocation of resources for a process. If we consider an operation like context switch which is very time consuming while switch to the different thread not bothering about saving of context. This can boost the performance with lesser cost.

Scalability. Threads are running simultaneously, when we have multi-processor architecture threads of a process can distribute among processors. This architecture increases parallelism and different tasks of a single process is executing together in the environment.

## 3.5ISSUES OF MULTICORE PROGRAMMING

Multithreaded programming uses multiple resources together and shared over the many threads and processes. Such multicore system require fine tune system design and also code developed by application developer that makes better use of multiple computing core. Today’s programmer has challenge to modify their conventional programming to take advantage of multicore system by using multithreaded programming. There are certain issues need to be taken care for the programmer in multicore system.

Divide load. Examine the application carefully and find out task that can be separate or further divide into sub task. This sub task is necessarily independent so it can run in the environment concurrently.

Balance. While programmer identify tasks, he/she must ensure that all the tasks perform equal work of equal values. To divide task of the process so each task has equal contribution in process execution.

Split the data. While divide the process into tasks, sharing of data will also take place; the access of data and manipulation of data through the tasks must be divided to run on separate core.

Dependency of data. When there is a data sharing, it is possible that data used by one instance of task also used by another instance of the task in the same environment. Programmer has to ensure that such tasks are synchronized or not. Such phenomenon gives inconsistency in the system.

## 3.6MULTITHREADED MODELS

As shown in figure 3.1, thread is consist of kernel area and user area. Support for threads may be provided either from user area to user thread or kernel area to kernel thread. User thread are supported above kernel and managed without kernel support. While kernel threads are supported by operating system. Most of all contemporary operating system supports kernel threads.

Relationship between user threads and kernel threads are divided into three common ways as discuss here.

Many-to-One Model

This model maps many user level threads to one kernel level thread. Thread management is done by the thread library of user area. Problem of the system is, if one thread of the process is blocking system call than entire process will be blocked. Thus only one thread can access a kernel at a time, multiple thread are unable to run in the multiprocessors architecture. Usage of resource is not worth here.

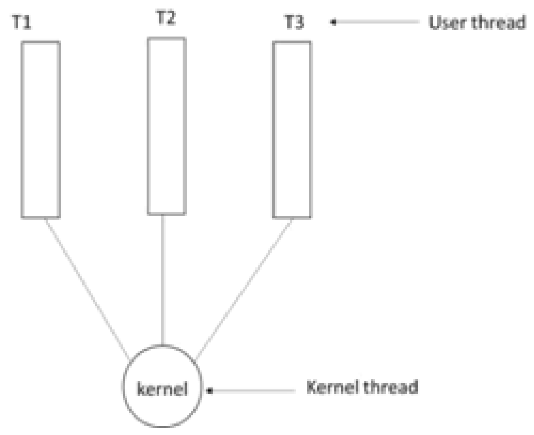


***Figure 3.2: Many-to-One Model***

One-to-One Model

This model maps each user thread to a kernel thread. This is more concurrent than previous model many-to-one model because it allows another thread to run in the environment when one thread is blocking a system call. A drawback of this model is when you create user thread it is also required that you must have kernel thread. More kernel thread creates extra burden to the performance of the system. Linux and its flavors as well as windows operating system are using one-to-one model of thread. Figure 3.3 shows one-to-one model.

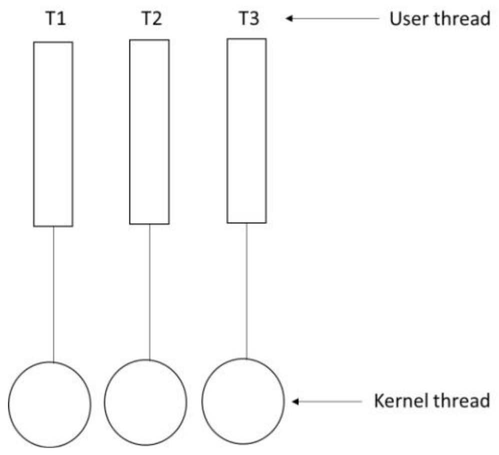


**Figure 3.3:One-to-One Model**

Many-to-Many thread Model

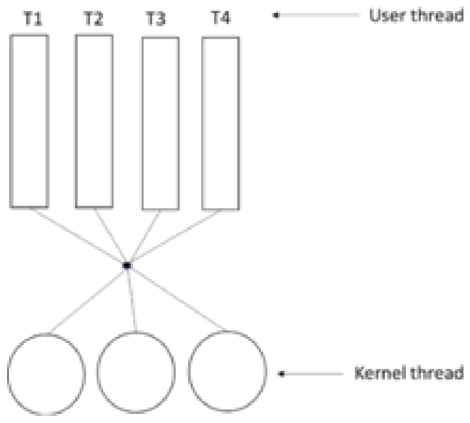


**Figure 3.4: Many-to-Many Model**

This model multiplexes many user-level threads to a smaller or equal number of kernel-level threads. This model allows the programmer to create as many user threads as they wants in the environment is shown in the figure 3.4. This model allows programmer to create as many user threads as necessary and provides best concurrency; when one of the thread is blocking system call, kernel can schedule another thread for running.

## 3.7PROGRAM OF THREADS

Here we are going to discuss program of thread using Java threads.

Multithreading in Java

Using multithreading feature of a java we can create concurrent execution of two or more parts of a program for maximum utilization of CPU resources. Each part is called a thread. In java threads can be created using two mechanisms – extending the Thread class and implementing the Runnable Interface. Here this program has extended the thread class. Prime and Even are two classes extends java.lang.Thread class. This class overrides the run() methods available in the Thread class. The thread begins its life through this method. After creating an object of this class we can call a method Start() to execute the thread. Start() implicitly invokes the run() method on the Thread object.

```
Class Prime extends Thread // declaration of
Prime class that extends Thread {
public void run()
{
for(int I = 1; I<= 200; i++)
{
double f = 1;
for(int j = 2; j <I; j++)
{
if(i%j == 0)
{
f = 0;
break;
}
}
if(f == 1)
{
system.out.println(“Prime: “+i);

}
try{
Thread.sleep(500);
//interrupting a thread
if(f==0)
{
this.interrupt();
}
}catch(IllegalArgumentException iae){
System.out.println(“”+iae.getMessage());
}catch(InterruptedException ie){
}
}
}
}
class Even extends Thread // declaration of Prime class that extends Thread
{
public Even()
{

}
public Even(String s)
{
super(s);
}
public void run()
{
this.setPriority(5);
```

```
for(int I = 2; I<= 100; I = i+2)
{
System.out.println("\t\t\t\tEven "+i);

try{
                                Thread.sleep(500);
                  }catch(IllegalArgumentException iae){
System.out.println(""+iae.getMessage());
}catch(InterruptedException ie){
System.out.println(""+ie.getMessage());
}

}
}
}
public class ThreadPractical
{
public static void main(String[] args)
{
Prime prime = new Prime(); // an object of Prime class
Even even = new Even("Even"); // an object of Even class
prime.setDaemon(true);

//Display Name of Threads
System.out.println("Name of Prime Thread Before Set a Name: "+prime.getName());


//Setting Priority and Name of thread
try{
prime.setPriority(Thread.MIN_PRIORITY);
prime.setName("Prime");
even.setPriority(Thread.NORM_PRIORITY);
System.out.println("\nPriority of "+prime.getName()+" Thread: "+prime.getPriority());
System.out.println("Priority of "+even.getName()+" Thread:
"+even.getPriority()); }catch(IllegalArgumentException ie){
System.out.println(""+ie.getMessage());
}catch(SecurityException se){
System.out.println(""+se.getMessage());
}


//Display Name of Threads
System.out.println("\nName of Prime Thread After Set a Name: "+prime.getName());
```

```
//check if thread is alive

if(prime.isAlive())
{
System.out.println("\nThread prime is Alive. ");
}


else
{
System.out.println("\nThread prime is not Alive. ");
}

//Starting a thread
try{
prime.start();
//count no of threads before starting even thread
System.out.println("\nCurrently Active Threads(Before Even Start):
"+Thread.activeCount()+"\n");
//check if thread is alive after starting a thread
if(prime.isAlive())
{

                        System.out.println("\nThread prime is Alive. \n");
}
System.out.println("Prime\t\t\t\tEven");
prime.join(2000); //waiting for the thread prime to be died in 2000
ms even.start();
//count no of threads after starting even thread
System.out.println("\nCurrently Active Threads(After Even Start):
"+Thread.activeCount()+"\n"); }catch(IllegalThreadStateException e){
System.out.println(""+e.getMessage());
}catch(InterruptedException ie){
System.out.println(""+ie.getMessage());
}

//check access to modify by current thread
try{
prime.checkAccess();
System.out.println("\nThread "+prime.getName()+" can Modified By
Thread "+Thread.currentThread()+"\n");
}catch(SecurityException se){
System.out.println(""+se.getMessage());
}
}
}
```

## 3.8LET US SUM UP

- What is threads and how it works
- Concurrent execution of a code through a program and its benefits
- Different model of multicore programming
- Program of thread using java

## 3.9 GLOSSARY

| | |
|---|---|
| CPU | Central Processing Unit |

## 3.10REFERENCES

1) Operating Systems Concepts. – Addision – Wesley by Silberschetz A and Galvin.
2) Operating Systems-Design and Implementation – Prentice Hall of India Pvt. Ltd. By Tanenbaum and Albert S. Woodhull.
3) Computer Architecture – Pearson Education by Morris Mano
4) Modern Operating Systems – PHI By Andrew S. Tanenbaum

4

# Unit 4: Deadlock

## Unit Structure

4.1. Learning Objectives

4.2. Introduction

4.3. Criteria for Deadlocks

4.4. Deadlock principles

4.5. Deadlock prevention

4.6. Deadlock Avoidance

4.7. Detection and Recovery of Deadlock

4.8. Let’s sum up

4.9. Glossary

4.10. References

## 4.1 LEARNING OBJECTIVE

- Concurrent processes in the environment and usage of resources
- Situation that creates deadlocks
- Different methods for preventing or avoiding deadlocks in the system

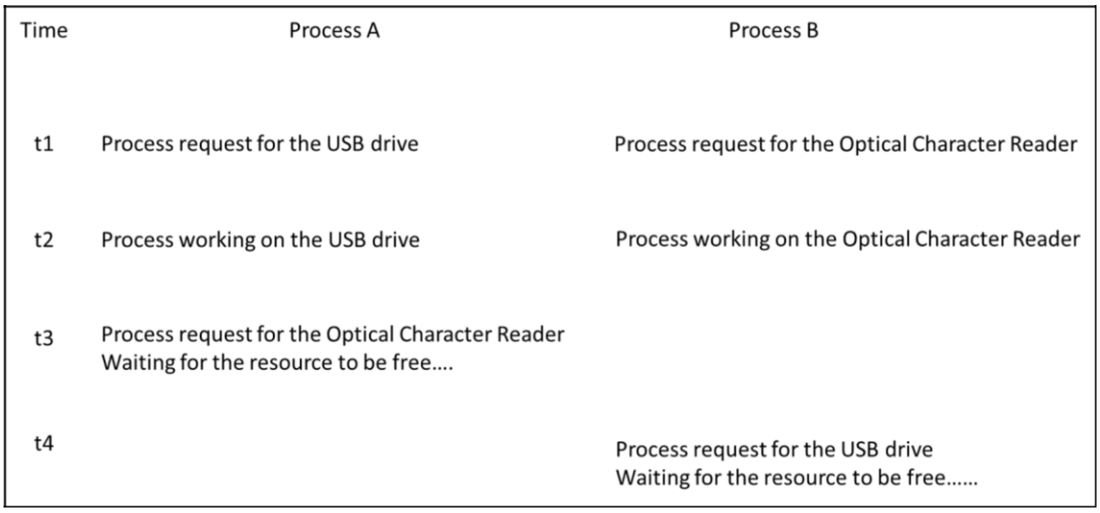
## 4.2 INTRODUCTION

In multiprogramming environment users may run several processes that take many resources in use simultaneously. When a process requests for some resource in this environment it is possible that required resource is already used by another process and that process is also waiting for some resource to be freed in the environment. Such situation creates deadlocks in the system. This chapter describes about situation when and when deadlock can occurs. Operating system apply many methods to deal with deadlocks and possibly try prevent situation. Generally operating system do not provide any facilities to prevent deadlock, it is programmers duty to write a code so that it can ensure the environment to be free deadlock.

## 4.3 CRITERIA FOR DEADLOCKS

Computer system having many resources that can used by many processes at a time. For example a process wants to print some data on a printer at the same time another process wants to write in the same data blocks. When both processes work on same resources create problem in access of the resource.

Now let’s understand the above example given in the figure 4.1; at given time t1, process A need to work on the data of USB drive. It request for the drive and get control of the drive. Same time process B request for the Optical Character Reader (OCR) and get control of the device and continue its work further using that device. At time t3, process A request for the OCR which is already used by process B. Here process A will wait for the OCR become free and rest in the sleep mode. At time t4, process B request for the USB drive. This drive is accessed by process A and it is in sleep because resource required by that process is locked by this process. Both the processes waiting for resources to be unlocked by each other. This is called deadlock in the system and it is not cleared by system itself.



| Time | Process A | Process B |
|---|---|---|
| t1 | Process request for the USB drive | Process request for the Optical Character Reader |
| t2 | Process working on the USB drive | Process working on the Optical Character Reader |
| t3 | Process request for the Optical Character Reader<br>Waiting for the resource to be free.... | |
| t4 | | Process request for the USB drive<br>Waiting for the resource to be free...... |

**Figure 4.1: Allocation of Resource and Deadlock Generation**

## 4.4 DEADLOCK PRINCIPLES

A set of processes is deadlocked if each process in the set is waiting for an event that is created by another process in the set.

All processes in the environment is waiting, none of them will ever cause any of event that creates wake up call for any process in the set, so all processes in the set are waiting state for infinity time.

Conditions for Deadlock

According to Coffman et al. (1971) described four conditions that locked resources simultaneously and creates deadlock in the system.

1. Mutual Exclusion Condition. Each resource is either assign to one and only one process or it is available. If another process requests that resource, the requesting process must be delayed until lock on the resource is released.

2. Hold and wait condition. Processes locking resources and waiting for the resources to be locked in advance for other purpose; that resources are locked by other processes.

3. No preemption condition. Resources that are granted to the process in past cannot be preempted forcibly until the process itself explicitly releases the lock of that resource.

4. Circular wait condition. In the system, if there are n processes going on; for example p0, p1,

p2….pn. If p0 is waiting for the resource locked by p1, p1 is waiting for the resource locked by p2 and on. At last process pn is waiting for the resource locked by process p0. This circular chain creates problem of deadlock in the system.

Activity I

What is called deadlock? Why it affect the performance of the system explain with an example.

## 4.5 DEADLOCK PREVENTION

The deadlock strategy is to impose restriction on processes in way that it is implicitly impossible to get deadlock. As per stated above four conditions are providing some clue to the solution of such situation in the environment. Let's take each condition one be one.

Mutual exclusion problem can be avoided by not allowing a resource exclusively assigned to a process. This will prevent the deadlock in the system. For example, if two or more processes writing on the hard disc at the same time will create chaos in the system. By sharable I/O task, many processes can do the I/O at same time.

All devices are not sharable in the system, for example a process table cannot be shared, so one process is working with process table other process cannot access data of the table and even can't able to read data from other slot in memory. Furthermore, if a process want to read data from the OCR and store it on the hard disc, after that it waits to sort the data and print it. Suppose sharable I/O task tried to write data before data has been collected from OCR then this process will wait indefinitely in the system. Similarly acquiring printer for the same task also wait and become idle for long time. This is why printer daemon are programmed to print only after the complete data is available.

Now considering the same example for the next condition hold and wait, there are two protocols suggested for the prevention. One is to acquire lock on all resources before you start your process. In this example we need three devices OCR, hard disc and printer. Initially we lock the OCR and complete the read task. Meanwhile intensive lock will remain on hard disc and printer. Once OCR task is over we release the lock on OCR and work with hard disc and then printer. If we can prevent processes that lock resources then we can eliminate deadlock. Other method allows the process to acquire a lock only when it is needed. So in the first task it lock the OCR and hard disc and write data on the disc. After that it will release the lock. Again to print the data, it will again acquire a lock on disc and printer and complete the task of printing data.

Third condition no preemption is very less promising, suppose a task of printing a file is going on in the system and meanwhile other process want resource to print. This time process is in middle of the printing task and forcibly taking away the printer because another process is needed the resource will make a system complex.

Last condition is circular wait, this can be eliminated by many ways. One is to provide a rule for a process is entitled to a single resource at a time. If a process has already locked a resource, it must be released first than it can acquire another resource in the environment. In this scenario, if a process demand a disc and printer to print a very large file than such restriction is not acceptable, both resources need to be available when data get printed by printer.

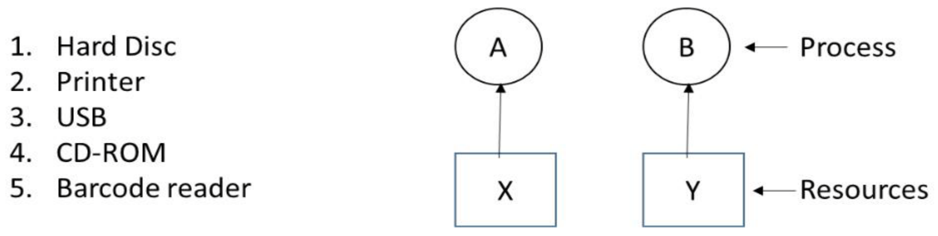


**Figure 4.2: Numerically ordered resources and resource allocation graph**

Other rule for this condition is to provide a global numbering system to all resources in the system. For example, in figure 4.2 shows numbering of resources and processes A and B has acquired resources X and Y respectively. According to this rule processes can request resources whenever it is required, this request can be serve on the base of number order given in the system. A process have a lock on lower numbered resource than it can acquired a lock on higher numbered resource. Vice a versa it is not true, if higher numbered resource is locked by a process than lower numbered resources are not acquired by the process. For example, as shown in the figure 4.2 hard disc got 1 number and Barcode reader got 5 number. If user is using hard disc and mean while it can request for the CD-ROM, this request is granted by the system but if a process is working with USB and want to print data than lock will not acquire by the process because printer has lower number than USB. Let us see how this rule prevent the deadlock situation. We get deadlock if process A request for the resource Y and process B request for the resource X; here X and Y are distinct resources. Suppose X > Y, then process A wants access of resource Y; system will allow to access that resource but if process B wants to access resource Y, it won’t allowed by the system. Either way, system will not have any deadlock.

If we consider a scenario with multiple processes, same logic will apply to the processes in the system. At a time one of the resource with given number is acquired by a process and it will never ask for the same resource that is already assigned. It will either finish or at the worst case it may request for the resource that has number lower than current resource. Eventually, it will finish and free its resources. Thus system will not have any deadlock.

When a process requested for the resources 4 and 5 number respectively and then releasing both in the system, it is effectively starting all over. There is no point to prohibit it now for requesting resource 1. Given numerical order of the resource eliminates the problem of deadlock but no number order will satisfies everyone. When resources like slot of process table, disk head, print spooler, locked database records and many more different uses will be very large that no ordering could possibly work.

Activity II

a) In what conditions, deadlock of the system can be prevented?

b) What do you think about No preemption? Why it is less promising?

## 4.6 DEADLOCK AVOIDANCE

In the figure 4.1, we saw that deadlock is avoided not by imposing arbitrary rules on the processes but careful analysis of each request and safely grant in the system. In such circumstances a question arises: Is there any algorithm that avoid deadlock by careful analysis? The answer of the question is yes, if certain information is available in the advance. Here we will learn about different ways by careful resource allocation.

### Banker’s Algorithm

In 1965, Dijkstra given an algorithm that can solve the problem of deadlocks and it is known as banker’s algorithm because it is modeled for the small-town bankers. In the small town bankers need to deal with customers for transferring credit, however very few customers are there. Bankers knows that not all customers need credit up to their maximum limits. In the figure 4.3, we have name of customers and their used credit units and maximum credit units. Banker have only 10 units as credits reserved and there are 20 units for the requirement service according to their maximum limit.

| Name | Used credit | Maxi. credit |
|---|---|---|
| Rambhai | 0 | 5 |
| Iftarbhai | 0 | 6 |
| Saluben | 0 | 4 |
| Kanji | 0 | 5 |

(a)

| Name | Used credit | Maxi. credit |
|---|---|---|
| Rambhai | 1 | 5 |
| Iftarbhai | 2 | 6 |
| Saluben | 1 | 4 |
| Kanji | 4 | 6 |

(b)

| Name | Used credit | Maxi. credit |
|---|---|---|
| Rambhai | 1 | 5 |
| Iftarbhai | 2 | 6 |
| Saluben | 2 | 4 |
| Kanji | 4 | 6 |

(c)

**Figure 4.3: Resource allocation in town with different demand of credit**

The customers go for their respective business and make loan request time to time. At particular moment of time as shown in figure 4.3(b), customers has used some of the credits. According to reserved credits of 10 units 8 units as been used by customers and 2 units are left. Here we can think that bankers is running in safe mode because if there exists a sequence of other circumstances that leads to all the customers will get loans up to their credit limits. Here the banker can delay any loan except Kanji's, thus letting Kanji to finish his loan and release all five credit points so other customers can get loan whenever they demand.

If we consider another scenario according to figure 4.3I, this is unsafe circumstances for the banker. Here he/she has only one unit left from the reserved units. If more customers ask for the loan up to their maximum unit, he/she cannot satisfy any of the customers. This will cerates deadlock, an unsafe state (In this analogy, customers are processes and credit units are like hard disc and banker is the operating system itself). Banker's algorithm will use to solve such problem. Let consider following data structure, where *n* is the number of processes in the system and *m* is number of resources in the system.

**Available.**A vector of length*m*indicates number of available resources. If*Available[j]*is equal to*k*, then*k* instances of resource type *R* are now working in the system.

**Max.** An*n * m*matrix defines maximum demand of each process. If*Max[i][j]*is equal to*k*then, supposeprocess *Pi*, may request at most *k* instances of resource type *R*j.

**Allocation.**An*n * m*matrix defines number of resources currently allocated to each process. If*Allocation[i][j]* equals k then process *Pi* is currently having *k* allocated instances of resource type *Rj*.

**Need.** An*n * m*matrix indicates the remaining resource need of each process. If*Need[i][j]*equal to*k*then process *Pi*, may need *k* more instances of resource type *Rj* to finish the task. Here *Need[i][j] =Max[i][j] – Allocation[i][j]*.

To check that whether system is running in the safe state or not, given algorithm will help us to find the state in the system.

1. Let *Work* and *Finish*are two different vector of length *n* and *m* respectively. Initially *work =Available* and *Finish[i] = False for I = 0, 1, 2… m-1.*
2. Find index *I* so both *Finish[i]* is *False* and *Need[i]* is less then equal to *Work*.

3. *Work = Work + Allocation[i], Finish[i]* is *True* and go to step 2.

4. If *Finish[i]* is True for all *I*, then the system is in safe state.

Resource request algorithm for determining whether request can be safely granted or not.

Let *Req[i]* is the vector of the process pi. If *Req[j]* is equal to k then process *Pi* want to access *k* instances of resource *Rj*. When request for resources is made by process *Pi*, then following are steps

1. If *Req[i]* is less than equal to *Need[i]*, go to step 2; otherwise give an error.

2. If *Req[i]* is less than *Avaialble* then go to step 3; otherwise process *Pi* must wait till resource is available.

*3.* Let system pretend to have allocated the requested resources to process *Pi* by modifying the state by *Availabe = Avaialabe–Req[i]; Allocation[i] = Allocation[i] + Req[i]; Need[i] = Need[i]–Req[i].*

Activity III

a) In figure 4.3 (b), Saluben has used 1 unit of credit, instead of 1 if she has used 2 units then the system will still remain in safe state? Why?

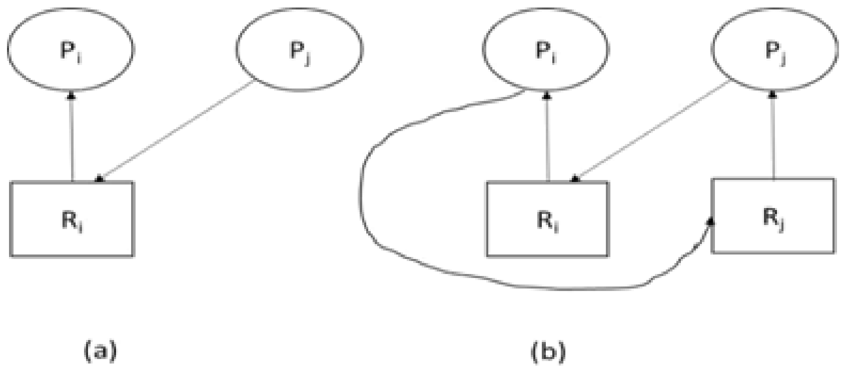
b) Many time a bus has single door to pick up passengers and drop passengers. It creates a deadlock when many people want to come out and same time many want to climb up. State a simple rule for avoiding deadlock in this system.

## 4.7DETECTION AND RECOVERY OF DEADLOCK

Sometime system does not provide either deadlock prevention algorithm or deadlock avoidance algorithm. In this environment, system may have an algorithm to determine whether a deadlock in the system is occurred or not. It also provides an algorithm for recovery from the deadlock. Let’s see first that how to detect deadlock in the system

**Deadlock detection.**

If all resources have a one and only one instance, deadlock-detection algorithm can be defined. Figure 4.4 shows the resources and processes allocation in the system. A process *Pi* acquired a lock on the resource *Rj* when it is working with the resource *Rj*. An edge from Pi -> Pj implies that process Pi is waiting for the resource that is acquired by the process Pj. Suppose, Rj is the resource, then Pi -> Rj and Rj -> Pj.



**4.4: Resource-allocation and wait-for graph**

A deadlock can exists in the system if and only if the wait-for graph is having a cycle. To detect the deadlock, the system need to check wait-for graph and invoke an algorithm that detect deadlock in the system.

**Deadlock Recovery**

When a deadlock detection algorithm determines a deadlock in the system, we can manually deal with the deadlock and remove it by killing the process. If deadlock is acquired by many processes then it is hard to find which process has really cause the deadlock. There are two approaches for the deadlock recovery.

Kill all deadlocked processes. This method will kill all processes at a time, so it is sure that deadlock will be removed but same time it is very costly. Those processes that may have computed for long time and get removed from the environment, all changes are discarded and need to be compute again.

Kill one by one process until cycle is eliminated. If cycle exists in the system, this method will kill one process in the environment. If this doesn't break the deadlock, we need to select another arbitrary process to kill and so on until the cycle is broken.

Activity IV

Define your algorithm for detecting deadlock for the scenario given in the activity III.(b).

## 4.8 LET US SUM UP

- Program of thread using javaWhat is deadlock and criteria for the deadlock
- Conditions for the deadlock prevention
- Deadlock avoidance and Bankers algorithm for avoidance of deadlock
- Deadlock detection and recovery

## 4.9 GLOSSARY

| | |
|---|---|
| OCR | Optical Character Reader |

## 4.10REFERENCES

1) Operating Systems Concepts. – Addision – Wesley by Silberschetz A and Galvin.

2) Operating Systems-Design and Implementation – Prentice Hall of India Pvt. Ltd. By Tanenbaum and Albert S. Woodhull.

3) Computer Architecture – Pearson Education by Morris Mano

4) Modern Operating Systems – PHI By Andrew S. Tanenbaum

# Block-2

# Memory and File Management

# Unit 1: Memory Concepts 1

## Unit Structure

1.1. Learning Objectives
1.2. Introduction
1.3. Logical and physical address
1.4. Swapping
1.5. Contiguous Memory Allocation
1.6. Let us sum up
1.7. Check your Progress
1.8. Check your Progress: Possible Answers
1.9. Further Reading
1.10. Assignments

## 1.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Understand the basic concepts of memory management like addressing
- Swapping mechanism
- Different algorithm related to contiguous memory allocation.

## 1.2 INTRODUCTION

In this unit, we are going to study about fundamentals of memory management like Physical Memory, virtual Memory and different algorithms of contiguous Memory Allocation schemes like single partition, fixed partition and dynamic partition.

## 1.3 LOGICAL AND PHYSICAL ADDRESS

- **Logical address** is generated by CPU. It is a virtual address, which does not exist physically on the main memory, so it is also called virtual address. The logical address space is mapped to physical address space. It is a reference to the physical address space.
- **Physical address** is the physical address of location on the main memory. The physical address is calculated for the corresponding logical address. MMU also uses logical address computing physical address. The user never deals with the physical address.

## 1.4 SWAPPING

Swapping is one of the important processes of operating system. It enables to manipulate files and execute programs larger then main memory. The page swap from main memory to secondary memory is called swap out and the page swap from secondary memory to main memory is called swap in. The swapping process allows executing multiple processes in parallel that improves the performance.

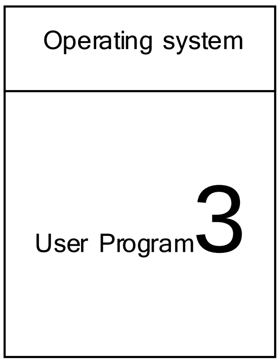
Swapping has three main parts: 1) Managing the swap space on swap device: 2) Swapping process out of memory 3) Swapping process into main memory.

The major time required for swapping is data transfer time from main memory to secondary memory and vice versa. The total time is directly proportional to amount of memory swapped, the processes around 10 MB can be transferred in less than 300 milliseconds and large processes about 250 MB can be transferred in about 7 seconds.

## 1.5 CONTIGUOUS MEMORY ALLOCATION

The operating system and the user processes are served by the main memory so the main memory is divided into two partitions. The one partition is for the operating system and the other is used for user processes.

**Single Partition Contiguous Allocation**



There are many memory allocation schemes. In the first scheme the complete program is loaded in the main memory in contiguous manner. The main limitation of this method is the size of program and main memory. If the size of program is higher than the size of main memory then the program cannot be loaded and executed.

The solution if this problem is either the size of the main memory should be increased or the size of the program should be altered.

The next limitation of this method is only a single process can be executed in the main memory and the remaining process must to wait until the running process completes. This means that this scheme does not support multiprogramming. The multiprogramming depends on the number of the programs executing in the main memory at same time.

The operating system code should not be interrupted from the user processes as both are running in the main memory. The protection to the operating system code is provided by the limit registers and relocation.
If the logical address generated by the CPU is less than the limit register than to get the physical address on the main memory, only the value of the relocation register is added to the logical address. An addressing error is generated if the logical address is greater than the limit register.

**Algorithm for loading the job according to single user contiguous scheme**

```
Step 1: set memory location in base address
Step 2: set pc= address of first memory location. Where PC stands for Program
        Counter.
Step 3: Read the instruction and increment the PC.
Step 4: Verify that, is it last instruction?
            If yes then stop loading the program and move to step-6
            Else move to next step- step -5.
Step 5: Verify that, size of PC is greater than memory size?
            If yes then stop loading the program and move to step-6
            Else load the instruction in the memory and read the next
            Instruction and move to –step-3
Step 6: Stop.
```

## Fixed Partitions

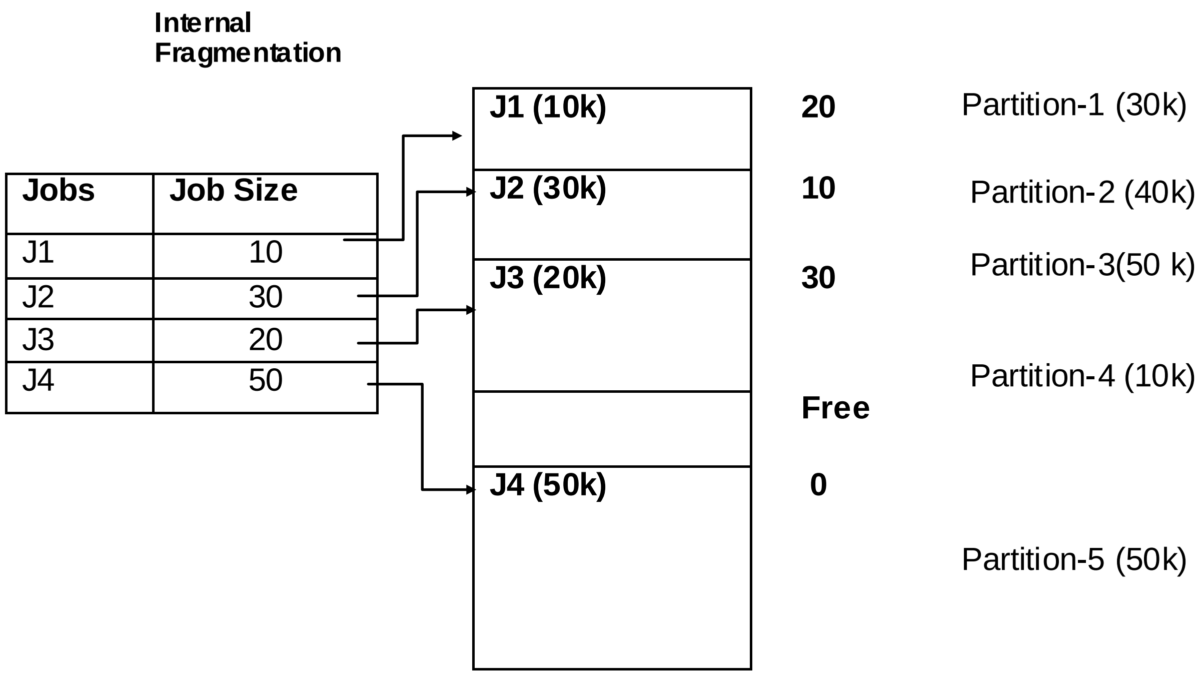
Multiprogramming is not supported by the single user contiguous scheme. To overcome the limitations, the concepts of static partitions or fixed partitions is used. In this type the main memory is divided into fixed sized partitions. The size of the partitions is created at the system start-up. Each partition can load one program. With use of this scheme multiple programs can be in the main memory at the same time. The problem of fixed partition is that operating system memory should be protected from the user program and each program in the main memory should be protected against other programs in other partitions. The programs may not cross the boundaries of partitions in any situations.

### Algorithm

```
Step 1: start
Step 2: Determine jobs requested memory size.
Step 3: identify that, job size is greater than size of largest partition?
            If yes than reject the job and move to Step-2.
            Else set counter =1 and move to Step-4.
Step 4: Identify that, counter is less than number of partitions?
            If yes than move to next step.Step-5.
Step 5: Identify if job size is greater than size of partition (counter)?
            If yes then move to next step.-step -6
            Else increase counter by one and move to step-4.
Step 6: Identify if partition status= “free”?
            If yes then move to next step. Step-7
            Else increase counter by one and move to step-4.
Step-7: load the job in the partition and set status= “busy”.
Step-8: if no partitions are available than put the job in the waiting queue.
Step-9: fetch the next job and move to step-2.
```

**Explanation of Algorithm**

This algorithm identify the size of the job to be executed and if the size of the job is higher than the biggest partition then the job will be rejected as there is not enough space in the memory to load the program. If the size of the program is less than the largest partition then set the value of counter as one. Here it will start from the first partition. Iteratively we will check that if the requested job size is less than the size of the first partition and the status of the partition is 'free' then load the job in the partition else increment the counter to the next partition. It should be verified till we reach to the last partition. If the job cannot be loaded in any of the partition but its size is less than the largest partition then it is placed in the waiting queue and new job is fetched.



| Jobs | Job Size |
|---|---|
| J1 | 10 |
| J2 | 30 |
| J3 | 20 |
| J4 | 50 |

As described in the example above if the job size is less than the partition size like job 1 (10k) is loaded in partition 1 of 30k, the rest 20k memory will be wasted and also now if Job5 of 20k arrives we cannot allocate the memory to it. Though there is 20k memory in partition -1 but it cannot be allocate to another job. It is known as **internal fragmentation.**

**First Fit Algorithm**

The first fit algorithm is fast search algorithm and search for the first sufficient partition from the top of main memory.

```
Algorithm:
Step 1: start
Step 2: set counter = 1
Step 3: verify that if counter is less then number of partitions?
        If yes then move to next step. Step-4
        Else verify that, job cannot be loaded put it in waiting queue and fetch
new
                Job and move to step -2.
Step 4: verify that if job size is greater than memory size(counter)?
        If yes then increase counter by one and move to step -3.
        Else move to next step. Step-5.
Step-5: load the job in the block and adjust free and busy list.
Step-6: fetch next job and move to step-2.
```

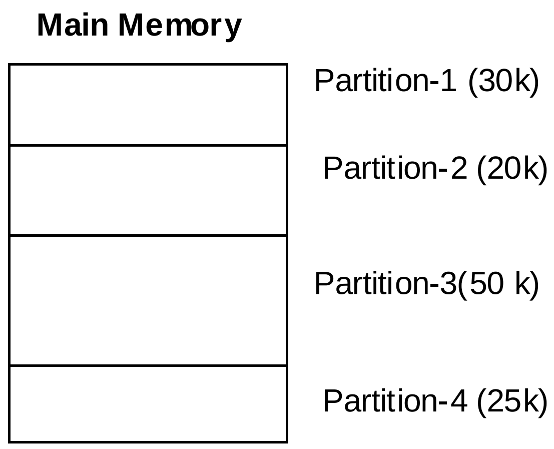
**Explanation of algorithm**

In this algorithm the first step is set the counter as one. Starting from the first partition, it checks iteratively to find that the job size is greater than the partition size and increase the counter. If the job size is less than the partition size then load the job in that partition and fetch the next job. After checking all the blocks if the job could not be loaded then put the job in the waiting queue and fetch the next block.

Consider a rack of books with many partitions. The librarian has to arrange the books in each partition. The librarian has to arrange the books in each partition. The librarian takes the book and checks on the first partition whether there is space available to put this book. If empty space is available then the librarian puts the book in this partition. If it is occupied the librarian checks the next partition till it reaches the end of the rack.

**BEFORE ALLOCATION**

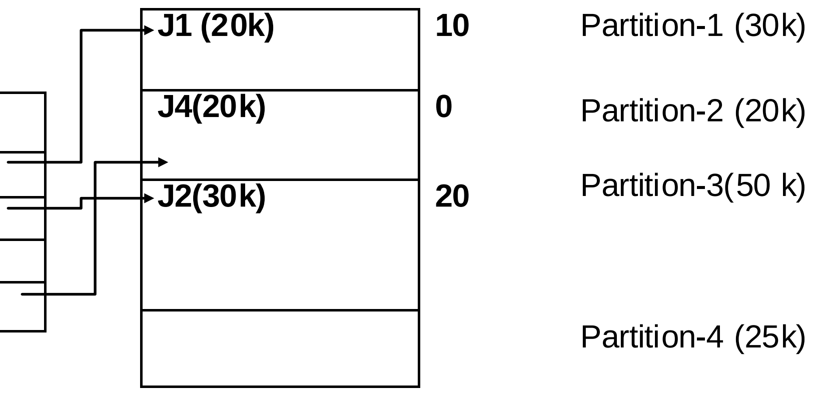
| Jobs | Job Size |
|---|---|
| J1 | 20 |
| J2 | 30 |
| J3 | 40 |
| J4 | 20 |



**AFTER ALLOCATION**

**Internal Fragmentation**

| Jobs | Job Size |
|---|---|
| J1 | 20 |
| J2 | 30 |
| J3 | 40 |
| J4 | 20 |



Here job j1 (20k) checks for partition -1 (30k), the job size is less than the partition size so we can load Job J1 in partition 1. For Job J2 (30k) , partition 1 is busy, for partition 2(20k) job size is greater than partition size we increment the counter to go to the next partition, for partition 3(50k) , job size(30k) is less than the partition size so J2 will be loaded in Partition-3, Job J3 (40k) checks for free partition 2 (20k) and partition 4 (25k), where it cannot be loaded so it is placed in the waiting queue. Job J4 checks for partition 2 (20k) first and it is loaded. This method is efficient in terms of processing and very less time consuming but it does not utilize the memory very efficiently.

**Best Fit Algorithm**

This algorithm searches the complete list of free partitions and allocates smallest partition which is sufficient for the requirement of the requested process.

```
Algorithm:
Step 1: start
Step 2: initialize partition zero. Size = maximum value.
Step 3: calculate the initial memory waste through deducting job size from size
            of partition zero.
Step 4: initialize Partition with minimum waste to zero. (PMW=0)
Step 5: verify that if counter is less than or equals to number of partitions?
            If yes then move to next step. Step -6.
            Else move to step- 8.
Step 6: job size is greater than partition size?
            If yes then increase COUNTER by one and move to step -5
            Else calculate memory waste by deducting job size from partition size
                and move to step -7.
Step 7: verify if initial waste is greater than current memory waste?
    If yes then make PMW= COUNTER and
            initial memory waste = current memory waste and increase
            the COUNTER by one and move to step-5.
Step 8: verify if PMW is equal to zero?
            If yes, then put the job in waiting queue and fetch the next job.
            Else load the job in the partition with minimum memory waste( PMW)
        and Fetch the next job and move to step-2.
```

**Explanation:**

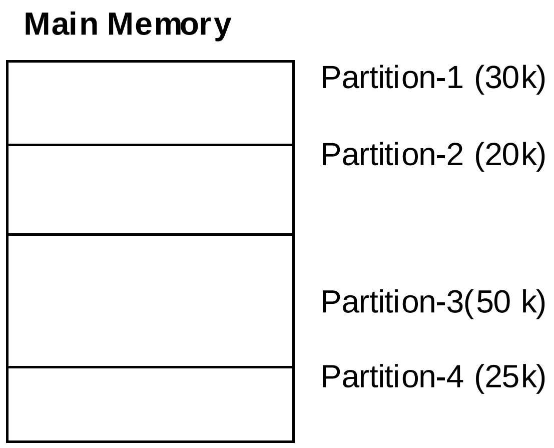
The memory partition-zero is assigned a maximum value. The initial memory waste is calculated through memory partition zero – job size. In this scheme the partitions are declared of variable size. The Partition with Minimum Waste (PMW) is declared that stores the value of the counter where it finds minimum wastage of memory. Initialize count =1 that is starting from first partition.

Check the value of counter till it reaches the total number of partitions in the memory. if the job size is less than the partition size the calculation of the memory waste is equal to partition size (counter) – job size. If this current memory waste is less than the previous memory waste then we assign the value of counter to the PW and initial memory waste = current memory waste and increment the counter**.**

If job size is larger than the partition size(counter) then move to next partition +and increase counter size by one. If all the partitions are verified but if the value of PMW=0 that means that the partitions are not free and put the job in the waiting queue. If PMW is non zero then load the job in that partition (PMW).

The example of Best fit algorithm is, mentioned here. The librarian takes the book and puts the book in each free rack, it verifies the wastage of space in each rack and find the rack where he finds the minimum waste in all the racks and then places the book in that rack. The benefit of this method is that it utilizes the memory very efficiently. The limitation of this method is to find minimum waste it increase the overhead on the processor and reduces the efficiency in terms of processing and it is time consuming.

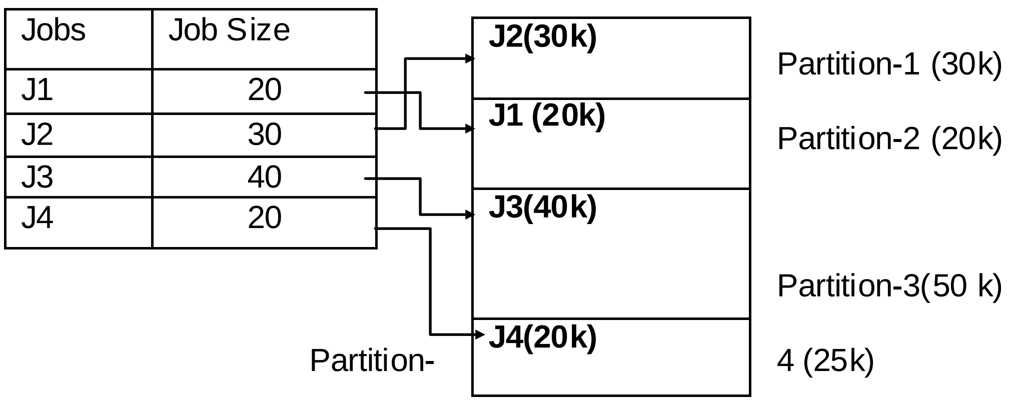
| Jobs | Job Size |
|---|---|
| J1 | 20 |
| J2 | 30 |
| J3 | 40 |
| J4 | 20 |



**Calculation of memory waste for job j1 (20k)**

The memory waste of each partition is calculated as Partition size – Job Size

In the first partition memory waste is 30-20 = 10K , in Second partition it is 20-20 =0k and in third partition it is 50-20 = 30k and in fourth partition it is 25-20 = 5k Here the minimum waste is in partition 2 ( 0 k) so job is placed in the second partition. Likewise all jobs are allocated to frames.



As compared to the example in First Fit we are not able to allocate the memory for Job J3 but according to best fit algorithm we can utilize the memory very efficiently and can allocate the memory to the jobs with less amount of internal fragmentation.

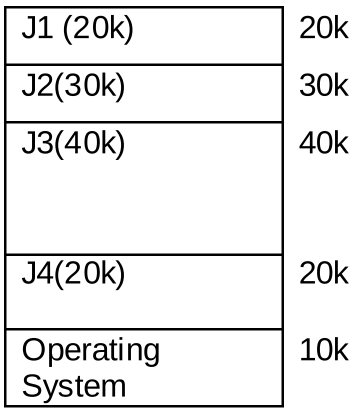
**Dynamic Partitions**

In dynamic partitions the jobs are allocated memory according to the size of the job. . The limitation of fixed partition scheme is internal fragmentation. To solve this problem partitions are created equal to the size of the job. For example, if the job J1 size is 20k then the partition will be created dynamically of 20k size and the internal fragmentation can be avoided.

Now imagine that there are four jobs J1, J2, J3 and J4

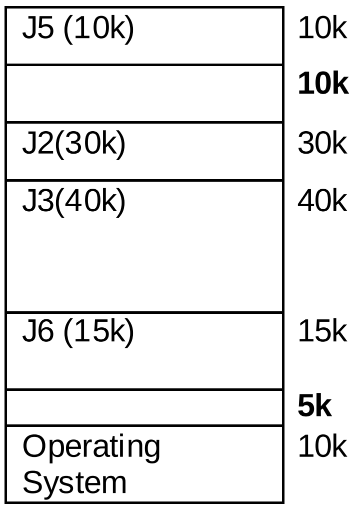
**Main Memory**

| Jobs | Job Size |
|---|---|
| J1 | 20 |
| J2 | 30 |
| J3 | 40 |
| J4 | 20 |



The jobs are allocated the memory blocks dynamically and the blocks created are equal to the size of the job so there is no internal fragmentation and memory is utilized efficiently.

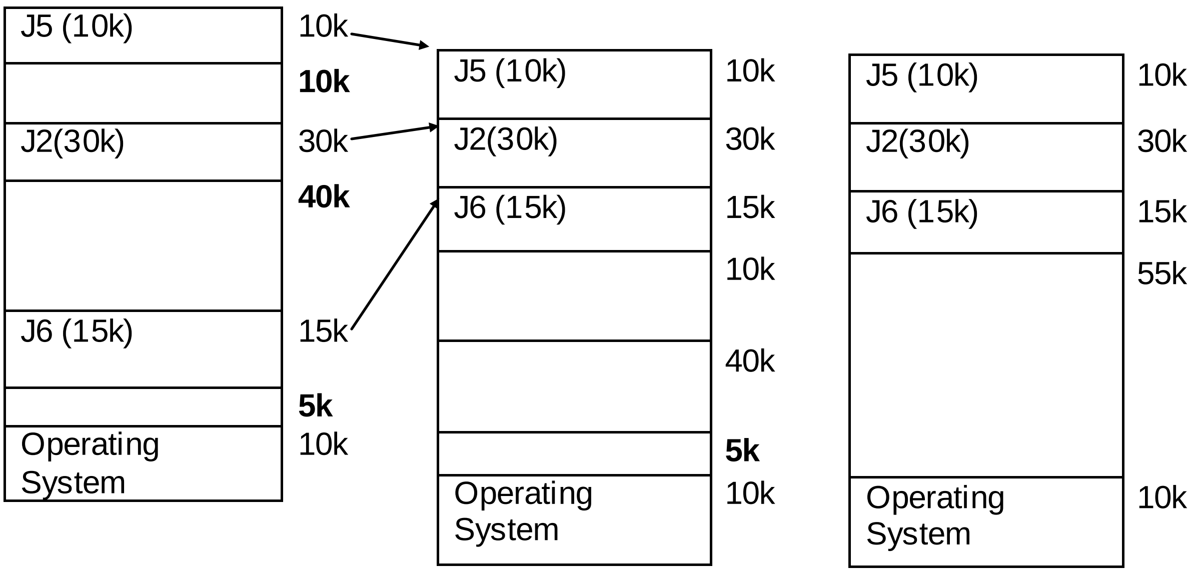
Now suppose Job j1 and J3 completes execution and rest of the processes are executing. So the memory becomes free and new jobs can be allocated this memory. Suppose Job J5(10k) and J6 (15k) arrives. They are allocated this memory.



Now suppose Job J7 (15k) arrives we are not able to allocate the memory to it. The reason is the memory is still fragmented. (Fragmentation means like breaking the mirror into pieces). The 5k memory block can be allocated to some new job J8 (2k) then there will be two blocks of 2k and 3k, still 3k can be assigned to 2k job, it will be further fragmented. At one stage the memory will be completely fragmented that it may not be able to load a single job. This is known as **external fragmentation.** It is a drawback of dynamic partitions. It removed internal fragmentation completely but it gave rise to external fragmentation problem.

**Relocatable Dynamic Partitions**

In dynamic partitions, the limitation is external fragmentation, to overcome this problemrelocatable dynamic partitions can be created. According relocatable dynamic partitions concept, after dynamic partitions external fragmentation is generated and the free partitions can be relocated and merged to create a larger free space. This free space can be utilized to load new jobs.



Suppose there is a colony with about 9 Apartments; suppose a person was living in apartment number 1 and now he has shifted to apartment number 9. A postman arrives but he has the old address on the letter to be delivered to that person. What the person did he placed a notice board near apartment 1, stating “add +8 to my old address to get my new address". So the postman can read the notice and deliver the post at the right address to the person.

Here the postman is the processor which has the old address of the process which was executing in the main memory. Now it has been relocated to a new address. The notice board here is the relocation register which stores the value to be added to the old address to get the new address of the process which was executing. One more register is used which is known as bounds register which would check the boundaries. If we add +10 to old address then we get 11, which is out of limit of the apartments. Similarly it checks whether the new address obtained is within the memory block range.

A relocation register contains the value that must be added to each address referenced by the CPU so that it will be able to access the correct memory location after relocation.

A bounds register is used to store the maximum limit of the memory locations which the user programs can access.

Now the question is that when this relocation should be carried out. There are three solutions for this. One we can relocate this partitions after fixed a fixed percentage of memory is occupied. (for ex: 70% of the memory is occupied). Second we can relocate after a specific interval of time (for ex: every 30 mins) and third we can relocated the partitions when any new job arrives and there is no free space to load this job.

If we relocate the partitions at shorter intervals then it increases the overhead on the CPU and large intervals then memory may not be utilized efficiently thus a proper selection may optimize the processing, utilization of memory and reduction in CPU overhead.

## 1.6 LET US SUM UP

In this unit, we have learnt about the contiguous memory allocations in single user contiguous scheme, fixed partition schemes and algorithms to overcome internal fragmentation. We also learnt dynamic partition schemes and its problem of external fragmentation and how relocation of dynamic partitions can solve it to certain extent.

## 1.7 CHECK YOUR PROGRESS

### True or False

1. Single user contiguous scheme can support multiprogramming
2. Dynamic memory allocation can remove the problem of fixed partitions
3. Relocation of dynamic partitions can increase the overhead on CPU
4. Best fit algorithm is more efficient than first fit algorithm
5. In single user contiguous scheme if the job size is larger than the memory size the job is rejected

### Fill in the blanks

1. ____ register is added to the logical address to get the relocated address.
2. _____ algorithm uses memory more efficiently in fixed partitions
3. _____ fragmentation occurs in fixed partitions
4. Internal fragmentation is calculated as partition_size - ______
5. In dynamic partitioning the size of the partitions are equal to _____

## 1.8 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

### True or False

1. Single user contiguous scheme can support multiprogramming (FALSE)
   It does not support multiprogramming as memory can be allocated to a single job
2. Dynamic memory allocation can remove the problem of fixed partitions (TRUE)

3. Relocation of dynamic partitions can increase the overhead on CPU (TRUE)
4. Best fit algorithm is more efficient than first fit algorithm (FALSE)
   Best fit algorithm utilizes the memory more efficiently but takes more time for processing
5. In single user contiguous scheme if the job size is larger than the memory size the job is rejected (TRUE)

**Fill in the blanks**

1. **RELOCATION** register is added to the logical address to get the relocated address.
2. **BEST FIT** algorithm uses memory more efficiently in fixed partitions
3. **INTERNAL** fragmentation occurs in fixed partitions
4. Internal fragmentation is calculated as partition_size – **JOB SIZE**
5. In dynamic partitioning the size of the partitions are equal to **JOB SIZE**

## 1.9 FURTHER READING

Books for further reading in above topics are:

(1) OperatingSystemsConcepts.Addision–WesleyBySilberschetzAandGalvin.
(2) OperatingSystemsPranticeHallof IndiaPvt. Ltd. ByTanenbaum.
(3) OperatingSystems.McGrawHillBookCo.ByMadnickS.&DonovanJ.J.

## 1.10 ASSIGNMENTS

1. Explain single user contiguous scheme
2. Explain fixed partition scheme
3. Explain difference between best fit and worst fit algorithms
4. Explain dynamic partition scheme
5. Explain relocatable dynamic partitions

# Unit 2: Paging and Segmentation

2

## Unit Structure

2.1. Learning Objectives

2.2. Introduction

2.3. Paged Memory Allocation ( Paging)

2.4. Segmented Memory Allocation ( Segmentation )

2.5. Segmentation with Paging

2.6. Let Us Sum Up

2.7. Check Your Progress

2.8. Check your Progress: Possible Answers

2.9. Further Reading

2.10. Assignments

2.11. Activities

2.12. Case Studies

## 2.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Understand the non-contiguous memory management concepts in much detail and how to allocate the memory frames to a program avoiding the external fragmentation.
- How paging and segmentation techniques helps to reduce the fragmentation problems and use memory effectively.

## 2.2 INTRODUCTION

In this unit, we are going to study about the non-contiguous memory management techniques like paging, segmentation and segmentation with paging. Here the basic idea is to reduce the fragmentation which is a major problem with fixed partition or dynamic partition schemes. These methods allow the pages or segments to be allocated to non-contiguous memory locations and also can keep track of those locations.

## 2.3 PAGED MEMORY ALLOCATION (PAGING)

Paging or paged memory allocation refers to allocation of physical memory to a program in non-contiguous manner while reducing the problem of internal fragmentation. The program is divided into equal sized parts called pages and memory is divided into equal sized partitions called frames. Generally the size of the page is equal to the size of the frame to avoid internal fragmentation. Now here the memory (frames) are not allocated sequentially.When the program has to be executed, thenumber of pages are determined. The number of available free frames are located and if there are enough frames available then the pages are loaded on this frames. It is not necessary that this frames are in contiguous sequence, so the operating system needs to keep track as which page of the program is allocated to which frame and to manage this it maintains a table called Page map Table (PMT). PMT is created for each program in the memory. It stores the page number and the frame number where the page is actually loaded in the main memory. Each program has its own page map table which is also stored at a memory location. The job number and location of page map table of that job is stored in Job Table. Another table is Memory Map Table (MMT) which keeps track of free and busy frames in the main memory.

New frames have to be allocated to a new job created, those frames which are free can be allocated to this job. Memory map table contains the frame numbers and its status of the frame i.e. free or busy. To locate a page of already executing job, first the program is located in the Job Table, from where we get the location of Page Map Table of the specific job. From the page map table we can search for the exact page number to get its corresponding frame number in the main memory.

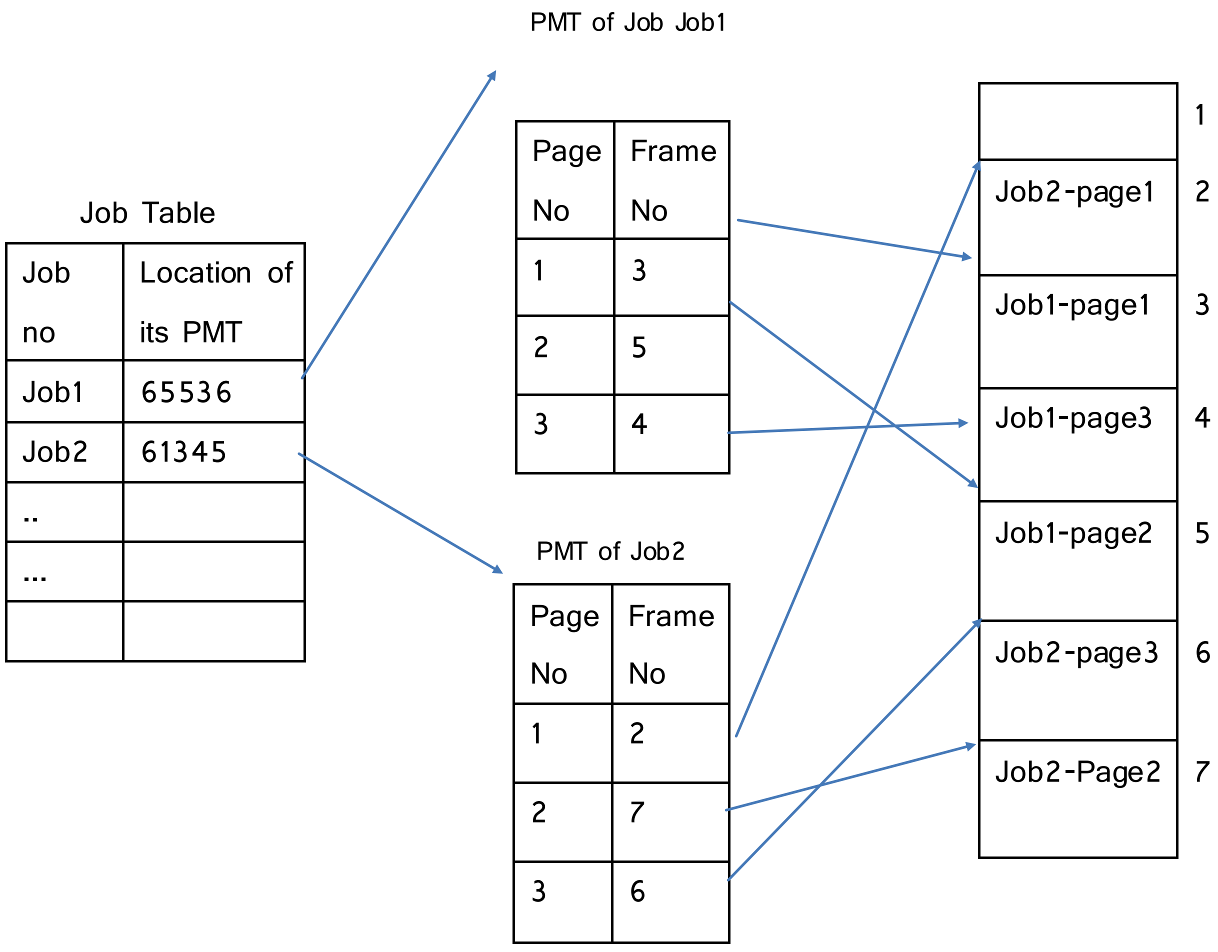


**Figure-1: Paged Memory Allocation**

The CPU generates the logical address containing the page number (p) and the offset (d) within the page. From the page map table, the frame is identified and the offset is added to the frame to get the physical address on the main memory.Example: If we want to access page 2 of job j1 with offset of 20k. Each frame is of 100k. The location of PMT for Job J1 = 65536. From this page map table for job j1 we get page number 2 and its corresponding frame that is frame 5 . Now multiply the size of the frame to it 5*100k = 500. Now add the offset to it. 500 + 20 = 520. It is the location of page 2 of job1 with displacement 20.

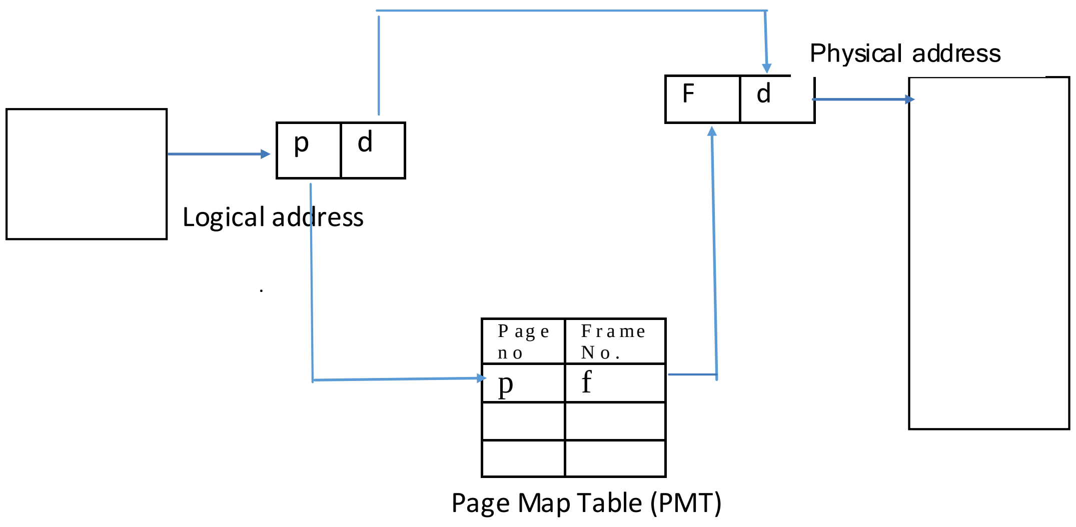


**Figure-2: Paging Hardware Assistance**

Address generated by CPU is divided into Page Number and Page Offset.

Page number (*p*) is used as an index into a *pagetable* which contains the physical address (frame number) of each page in the memory

Page offset (d) is combined with the physical address (frame number) to define the physical memory address

Pagenumber(p) Page offset (D)

| | |
|---|---|
| p | D |

## Translation Look-aside Buffer (TLB)

Translation Look-aside Buffer is a cache used to reduce the access time taken to search a page entry in the Page Map Table. TLB contains the page numbers and corresponding frame numbers of frequently used pages, so the time required to search the pages in PMT is reduced, if the page is found in TLB it is called TLB hit and if the page is not found, then it’s known as TLB miss. In the case of TLB miss, the page is searched in the Page Map Table. In case of TLB hit, it can get the corresponding frame number and adding the offset to get the physical address on the main memory quickly.

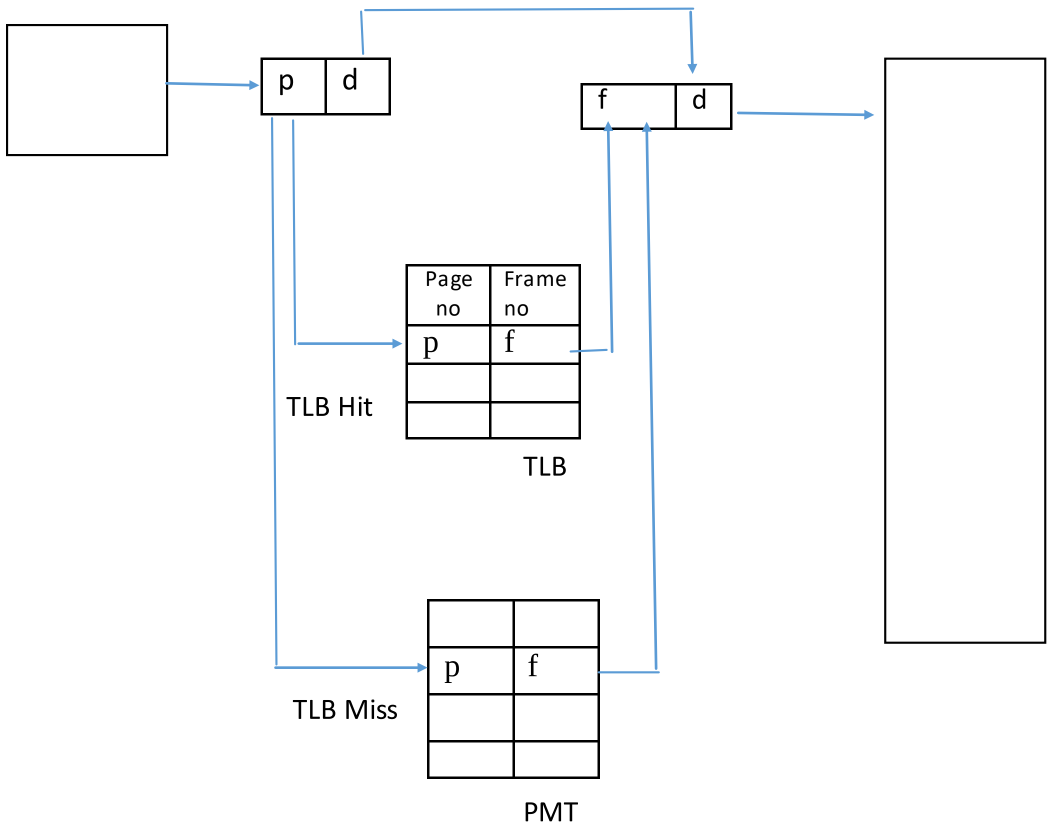


**Figure-3: Paging with Translation Look-Aside Buffer (TLB)**

**Memory Map Table (MMT)**

In case when a new job is created, new frames need to be assigned to the pages of the job. The memory map table keeps track of free and busy frames, busy means job is allocated to the frame, so the free frames are identified from the memory map table and are assigned to the pages of new job, and the status of this allotted frames is updated to busy.

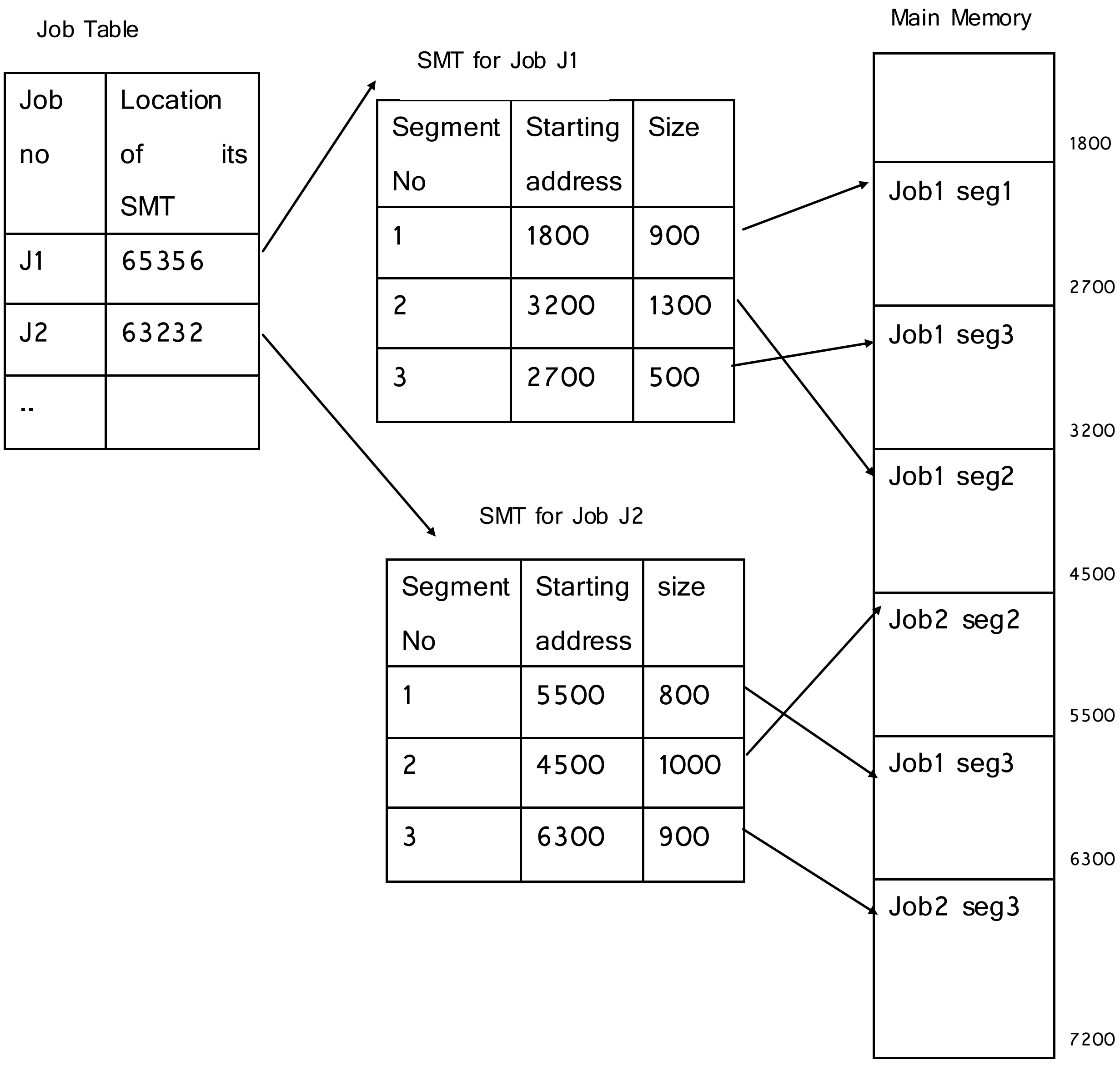
In paging, the drawback of internal fragmentation still persist in the last page. If the program is of 45k, the program is divided into 10k pages and frames are of 10k each. Four frames will have no internal fragmentation as frame size and page size are of 10K, but the last frame remaining will have 5k page and the frame size is 10k, thus last frame will have 5k memory wasted. Thus there is still internal fragmentation in the last frame.Paging is able to reduce the internal fragmentation up to the last frame but is not able to eliminate internal fragmentation.

## 2.4 SEGMENTED MEMORY ALLOCATION

Segmentation is based on the concept dividing programs into unequal sized logical parts rather than dividing into fixed sized pages as in paging. For example, let's divide the C language program into segments, we can divide it into segments like variables, functions, sub routines etc. This segments are loaded using dynamic memory allocation to eliminate internal fragmentation as in the case of paging. Here there are no fixed sized frames. This segments are loaded at different locations of the main memory and to keep track of the segments and its locations Segment Map Table (SMT) is used. SMT contains segment number, starting address or base address, size or limit of the segment, status, access information etc. Segment Map Table is unique for each job.

Here also Job table is used, which contains job number and location of its SMT in main memory. To locate a segment 3 of job number 1, the address of its Segment map Table is required. From Job table, job number 1 is located to get its corresponding address of its SMT. In SMT, the segment 3 is located to get its corresponding starting address and the size of the segment on the main memory, therefore ending address will be calculated as ending address = starting address +

segment size. Thus finally, the starting address and the ending address of the segment is located on the main memory. The Memory Map table keeps track of address locations which are free or busy, so when new job arrives, the segments are loaded dynamically and the free memory locations are allocated to the segments. The status of the allocated memory locations are updated to busy.



Job Table

| Job no | Location of its SMT |
|---|---|
| J1 | 65356 |
| J2 | 63232 |
| .. | |

SMT for Job J1

| Segment No | Starting address | Size |
|---|---|---|
| 1 | 1800 | 900 |
| 2 | 3200 | 1300 |
| 3 | 2700 | 500 |

SMT for Job J2

| Segment No | Starting address | size |
|---|---|---|
| 1 | 5500 | 800 |
| 2 | 4500 | 1000 |
| 3 | 6300 | 900 |

**Figure-4: Segmentation**

The displacement within a segment is added to the base address to access the specific location and as the segments are of unequal sizes, the displacement added to the base address should not exceed the segment limit.

For example if we want to access the Segment 2 of Job J1 with displacement of 200k. The location of SMT for Job J1 is searched from Job Table. Then after segment 2 is located from SMT to get starting address = 3200 and size of the segment is 1300k we get the ending address as 3200 + 1300 = 4500. The partition will be from 3200 to 4500. The displacement is added to the base address 3200 +100 = 3300. This displacement should be less than 4500 i.e. limit of this segment.

As this allocation scheme follows dynamic memory allocation, it still has the disadvantage of external fragmentation, for which relocation or compaction has to be done at optimal intervals.

## 2.5 SEGMENTATION WITH PAGING

The idea is of getting the advantage of both segmentation and paging and reducing the disadvantages of external fragmentation and long searching time of pages. Here the program is divided into segments and each segment is divided into pages. It differs from pure segmentation that the segment map table entry contains not the base address of the segment, but the address of a page map table forthis segment.

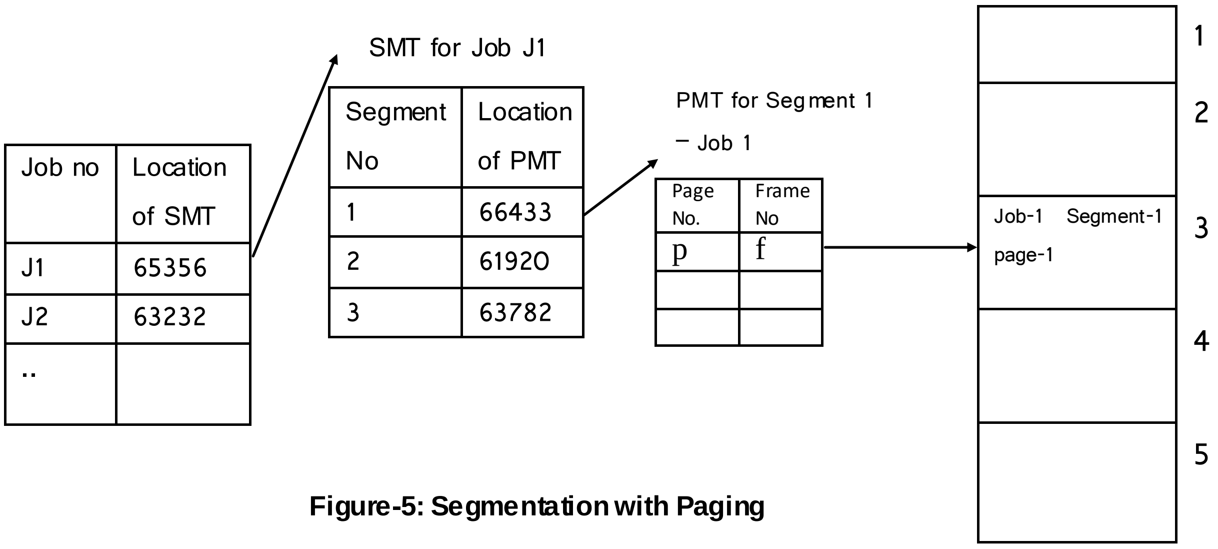


**Figure-5: Segmentation with Paging**

The page map table is created for each segment and it contains the page number and the corresponding frame number on the main memory.

The logical address contains the segment number, page number and the offset or displacement. The job is searched in the job table and from it the address of segment map table is located. The segment number (s) is searched from segment map table to get the address of the corresponding page map table. The page number (p) is searched from the page map table to get the frame number on the main memory and the offset (d) is added to get specific location.

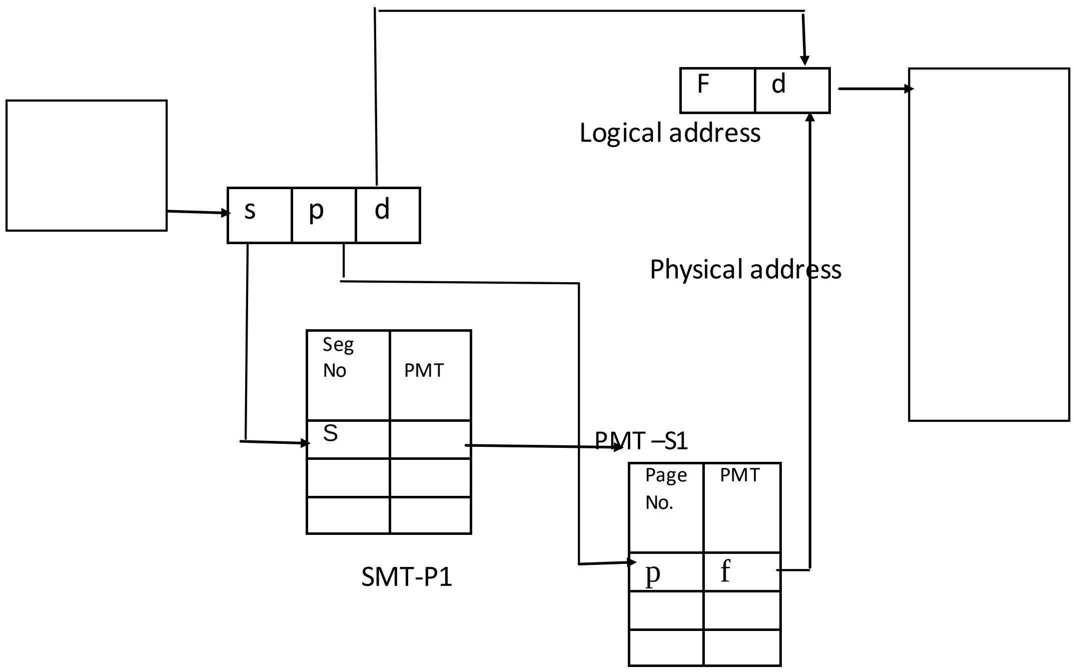


**Figure-6: Segmentation with Paging Hardware Assistance**

## 2.6LET US SUM UP

In this unit, we have learned about memory management techniques like paging, segmentation and segmentation with paging. This methods helps to reduce the fragmentation problems in the main memory and how to overcome the problems of memory management.

## 2.7 CHECK YOUR PROGRESS

### Fill in the blanks

1. The programs or jobs are divided into equal sized __________

2. The main memory is divided into equal sized ________
3. The programs are divided into unequal sized ________
4. In segmentation, the memory is allocated __________

### True or False

1. Internal fragmentation occurs in dynamic memory allocation
2. External fragmentation occurs in dynamic memory allocation
3. All pages are of equal sizes
4. The operating system maintains a page map table that keeps track of how many frames have been allocated or free.
5. Each job has a separate page map table in paging

## 2.8CHECK YOUR PROGRESS: POSSIBLE ANSWERS

1. The programs or jobs are divided into equal sized **PAGES**

   As it is mentioned equal sized it indicates that paging is used for allocation so the program is divided into pages.
2. The main memory is divided into equal sized **FRAMES**

   As again equal sized is mention so segmentation is not used and in paging memory is divided into equal sized frames.
3. The programs are divided into unequal sized **SEGMENTS**

   As unequal sized is mentioned it indicates segmentation and in segmentation the program is divided into unequal sized segments.
4. In segmentation, the memory is allocated **DYNAMICALLY**

   The memory is allocated as per the size of the segment and there are no fixed partitions in the memory.

### True or False

1. Internal fragmentation occurs in dynamic memory allocation (**FALSE**)

   Internal fragmentation only occurs in fixed sized partitions as in the case of paging
2. External fragmentation occurs in dynamic memory allocation (**TRUE)**
3. All pages are of equal sizes (**TRUE)**

4. The operating system maintains a page map table that keeps track of how many frames have been allocated or free. (**FALSE**)

   Memory map table is used to keep track of free and allocated pages.
5. Each job has a separate page map table in paging (**TRUE**)

## 2.9 FURTHER READING

Books for further reading in above topics are:

(4) OperatingSystemsConcepts.Addision–WesleyBySilberschetzAandGalvin.

(5) OperatingSystemsPranticeHallof IndiaPvt. Ltd. ByTanenbaum.

(6) OperatingSystems.McGrawHillBookCo.ByMadnickS.&DonovanJ.J.

## 2.10ASSIGNMENTS

1. Explain paged memory allocation with example
2. Explain segmented memory allocation with example
3. Explain Segmentation with paging and how it overcomes the limitations of paging and segmentation.
4. Explain advantages of Translation look aside buffer
5. Explain the difference between paging and segmentation.

## 2.11ACTIVITIES

Prepare a list of memory management techniques in different operating systems and try to identify the major advantages and disadvantages of the techniques used

## 2.12CASE STUDIES

Identify the memory management techniques of unix / linux operating systems and how they differ from each other

3

# Unit 3: Virtual Memory

## Unit Structure

3.1. Learning Objectives

3.2. Introduction

3.3. Demand Paging

3.4. Page Replacement Algorithms

3.5. Let us sum up

3.6. Check your Progress

3.7. Check your Progress: Possible Answers

3.8. Further Reading

3.9. Assignments

3.10. Activities

3.11. Case Studies

## 3.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Understand the concepts and benefits of virtual memory and how memory is allotted to the pages which are in demand
- Understand how page replacement techniques and which technique can be better for page replacements and improve the performance.

## 3.2 INTRODUCTION

In this unit, we are going to study about how pages are brought from secondary memory to primary memory and load the pages according to the requests generated by the CPU, but due to limitation of size of main memory, all pages of a program cannot be loaded into the main memory, so how to execute the program of size larger than the size of main memory is the focus of demand paging and page replacement policies.

## 3.3 DEMAND PAGING

In the previous chapters we have seen that to execute any program it has to be brought to the main memory for execution, there are two ways to do so. Either the entire program is loaded into the main memory or to load the program as needed. In paging and segmentation, the programs are either divided as pages or segments. The systems which uses paging architecture and if there is not enough physical memory to load all the pages can employ demand paging by swapping the pages between main memory and secondary memory. It also allows more number of processes to execute. To overcome the limitations of amount of physical memory, only the pages which are requested by CPU (in demand)would be loaded and rest of the program will be on the secondary memory. The pages which are not required are not loaded in the memory. This technique is called demand paging. For example machine having 512 megabytes of physical memory can execute a program of 1 gigabyte. The user feels as if whole program isexecuting in the main memory but actually only a part of the program is executing in the main memory and rest of the program is on the secondary memory. This concept isknown as virtual memory concept.The page which is in demand is brought to the main memory from the secondarymemory is known as swap-in. After it completes its execution and is sent from theprimary memory to the secondary memory is known as swap-out. The page which is indemand is not currently in the main memory, but it has to be brought from thesecondary memory, it is known as page-fault.

The data structures used by demand paging include

1) Valid bit: This bit is turned on by kernel if the page contents are legal

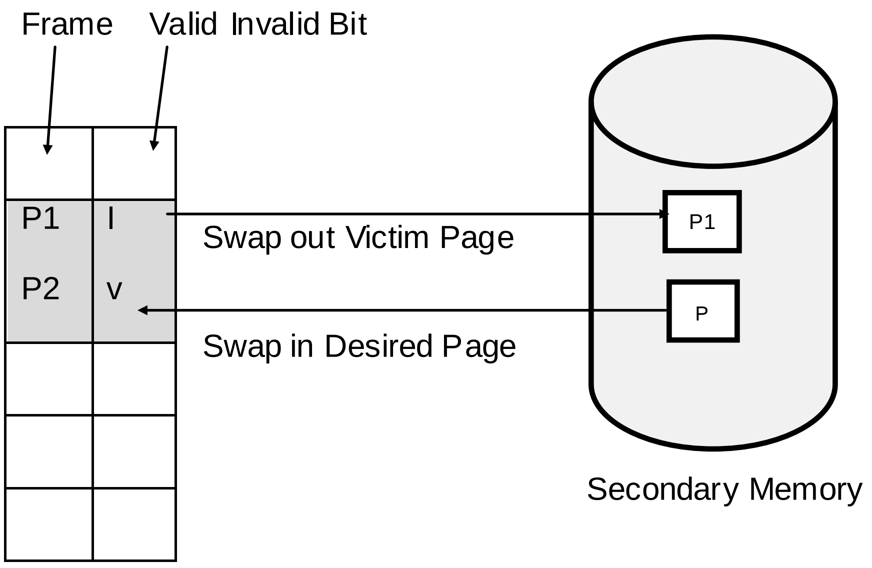
2) Reference bit: It indicates whether the page is recently referenced

3) Modify bit: It indicates whether the page is modified recently by a process.

4) Copy bit: It indicates a new page to be created when process modifies page contents.

5) Age bit: It indicates how long a page is in the working set of a process.

This data structures are generally used in UNIX systems



Main Memory

Here the valid-invalid bit is set to valid it means that associated page is valid and is in memory otherwise if it is set to invalid then the page is either not valid – the page is not in logical address space of the process  or it is valid but it is on secondary memory. In this situation a page fault trap is caused. The requested page is not in the main memory or failed to load the page into the main memory from secondary memory.

**Procedure for handling page faults**

We check for this process to determine whether the reference was a valid or aninvalid memory access.If the reference is invalid, we end the process. If it was valid, but not brought in the page in main memory, then find a free frame and schedule a disk operation to read the desired page into the newlyallocated frame.When the disk read is complete, the .page table updates that the page is now in memory and restarts the instruction that was interrupted by the page fault. The processcan now execute the page as it is in main memory.

The pages are allocated free frames available in the memory, but after that when frames are not available, and pages are requested by CPU for execution, any of allocated page has to be replaced. For example three students are sitting on three chairs and fourth student appears and has to be allotted a chair then one of the student has to leave the chair. Similarly one of the page from the main memory has to be replaced known as a victim page. The victim page is a page which is selected according to the page replacement policy.

## 3.4 PAGE REPLACEMENT ALGORITHMS

### First In First Out Replacement Policy

In this algorithm, the pages are allocated to the available page frames, as the number of frames available are not equal to the number of pages requested by the CPU, the pages have to be replaced. Now if there are three frames available and three pages are allocated to them and next page requested is not in the memory, it has to be brought in the memory but available frames are already allotted to the pages, therefore any of the three page has to be replaced. The question is which page should be replaced out of three. The first algorithm for that is First in First out. The page which was allotted the frame first has to be replaced by new page and this replacements continue for next pages. If the page is already in the memory then it is not brought from secondary memory to main memory.

The page which was requested by CPU is not available in the main memory, then it has to be brought from secondary memory to main memory. It is known as Page Fault.

Example: Calculate total number of page faults for the memory reference string:

7 0 1 2 0 3 1 4 2 1

With three page frames available using FIFO algorithm.

**Total page frames = 3**

| 7 | 0 | 1 | 2 | 0 | 3 | 1 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| **7** | 7 | 7 | **2** | | 2 | | 2 | | **1** |
| | **0** | 0 | 0 | | **3** | | 3 | | 3 |
| | | **1** | 1 | | 1 | | **4** | | 4 |

Total number of page faults: 7

Success percentage = (3/10) *100 = 30%

Failure Percentage = (7/10) *100 = 70%

Success: Failure ratio = 3:7

In the above example, the memory reference string is given and the number of page frames =3 are given. It means that three frames in the main memory are free where pages can be allocated. Initially all frames are empty, first page is requested by CPU is not in memory so first page 7 is brought in the main memory and allocated first frame, likewise 0 and 1 are allocated second and third frames. The fourth page 2 is requested is not in the memory so one of the pages 7, 0, 1 has to be replaced. According to FIFO algorithm, page 7 was allocated first so it has to be replaced by 2.

Now 0 is already in the memory so it is not replaced and it is not considered as page fault. Next page 3 is requested which is not in the memory, so page 0 will be replaced by 3. Similarly other pages are replaced in the memory and the page faults are calculated.

**Total page frames = 2**

| 7 | 0 | 1 | 2 | 0 | 3 | 1 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| **7** | 7 | **1** | 1 | **0** | 0 | **1** | 1 | **2** | 2 |
| | **0** | 0 | **2** | 2 | **3** | 3 | **4** | 4 | **1** |

Total number of page faults: 10

Success percentage = (0/10) *100 = 0%

Failure Percentage = (10/10) *100 = 100%

Success: Failure ratio = 0:10

**Total page frames = 4**

| 7 | 0 | 1 | 2 | 0 | 3 | 1 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| **7** | 7 | 7 | 7 | | **3** | | 3 | | |
| | **0** | 0 | 0 | | 0 | | **4** | | |
| | | **1** | 1 | | 1 | | 1 | | |
| | | | **2** | | 2 | | 2 | | |

Total number of page faults: 6

Success percentage = (4/10) *100 = 40%

Failure Percentage = (6/10) *100 = 60%

Success: Failure ratio = 4:6

In certain examples, as the number of page frame increases, the page fault also increases which is unexpected. This is known as Belady's Anomaly. If we give more frames (memory) the page faults should decrease.

## Least Recently used Replacement Policy

This algorithm is also used for page replacement. In this technique the least recently used page out of available pages on the page frames are replaced that means that the pages which are not used for longest period of time are replaced. The memory reference string is searched backwards and the page which is not used for longest time is selected for replacement from the page frames. The page which was allotted the frame first has to be replaced by new page and these replacements continue for next pages. If the page is already in the memory then it is not brought from secondary memory to main memory.

Example: Calculate total number of page faults for the memory reference string:

7 0 1 2 0 3 1 4 2 1

With three page frames available using FIFO algorithm.

Total page frames = 3

| 7 | 0 | 1 | 2 | 0 | 3 | 1 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| 7 | 7 | 7 | 2 | | 2 | 1 | 1 | 1 | |
| | 0 | 0 | 0 | | 0 | 0 | 4 | 4 | |
| | | 1 | 1 | | 3 | 3 | 3 | 2 | |

Total number of page faults: 8

Success percentage = (2/10) *100 = 20%

Failure Percentage = (8/10) *100 = 80%

Success: Failure ratio = 2:8

In the above example, the memory reference string is given and the number of page frames =3 are given. It means that three frames in the main memory are free where pages can be allocated. Initially all frames are empty, first page is requested by CPU is not in memory so first page 7 is brought in the main memory and allocated first frame, likewise 0 and 1 are allocated second and third frames.

| 7 | 0 | 1 | 2 |
|---|---|---|---|
| 7 | 7 | 7 | 2 |
| | 0 | 0 | 0 |
| | | 1 | 1 |

The fourth page 2 is requested is not in the memory so one of the pages 7, 0, 1 has to be replaced. According to LRU algorithm, in memory reference string before 2, the least recently used page out of 1,0, 7 is page 7, so it is replaced by 2.

| 7 | 0 | 1 | 2 | 0 | 3 |
|---|---|---|---|---|---|
| 7 | 7 | 7 | 2 | | 2 |
| | 0 | 0 | 0 | | 0 |
| | | 1 | 1 | | 3 |

Now, the page 0 is already in the main memory so it is not brought from the secondary memory to the main memory. Then page 3 is requested, so according to memory reference string before 3, pages 0, 2, 1 are requested, and out of them page 1 is least recently used so it has to be replaced by page 3. Similarly other pages are replaced in the memory and the page faults are calculated.

**Total page frames = 4**

| 7 | 0 | 1 | 2 | 0 | 3 | 1 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| 7 | 7 | 7 | 7 |  | 3 |  | 3 | 3 |  |
|  | 0 | 0 | 0 |  | 0 |  | 0 | 2 |  |
|  |  | 1 | 1 |  | 1 |  | 1 | 1 |  |
|  |  |  | 2 |  | 2 |  | 4 | 4 |  |

Total number of page faults: 7

Success percentage = (3/10) *100 = 30%

Failure Percentage = (7/10) *100 = 70%

Success: Failure ratio = 3:7

## Optimal Page Replacement Policy

According to this technique replace the page that will not be usedfor the longest period of time. The memory reference string is searched forward and the page which will not be used for longest time is selected for replacement from the page frames. The page which was allotted the frame first has to be replaced by new page and these replacements continue for next pages. If the page is already in the memory then it is not brought from secondary memory to main memory.

Example: Calculate total number of page faults for the memory reference string:

7 0 1 2 0 3 1 4 2 1

With three page frames available using FIFO algorithm.

Total page frames = 3

| 7 | 0 | 1 | 2 | 0 | 3 | 1 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| 7 | 7 | 7 | 2 | | 2 | | 2 | | |
| | 0 | 0 | 0 | | 3 | | 4 | | |
| | | 1 | 1 | | 1 | | 1 | | |

Total number of page faults: 6

Success percentage = (4/10) *100 = 40%

Failure Percentage = (6/10) *100 = 60%

Success: Failure ratio = 4:6

In the above example, the memory reference string is given and the number of page frames =3 are given. It means that three frames in the main memory are free where pages can be allocated. Initially all frames are empty, first page is requested by CPU is not in memory so first page 7 is brought in the main memory and allocated first frame, likewise 0 and 1 are allocated second and third frames.

| 7 | 0 | 1 | 2 | **0** | 3 | **1** | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| 7 | 7 | 7 | 2 | | 2 | | 2 | | |
| | 0 | 0 | 0 | | 3 | | 4 | | |
| | | 1 | 1 | | 1 | | 1 | | |

The fourth page 2 is requested is not in the memory so one of the pages 7, 0, 1 has to be replaced. According to Optimal algorithm, looking forward in memory reference string page 7 has no reference, page 0 has 2nd reference and page 1 has 4th reference, so page 7 is replaced by 2.

| 7 | 0 | 1 | 2 | 0 | 3 | 1 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| 7 | 7 | 7 | 2 | | 2 | | 2 | | |
| | 0 | 0 | 0 | | 3 | | 4 | | |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 1 | | 1 | | 1 | | |

Now, the page 0 is already in the main memory so it is not brought from the secondary memory to the main memory. Then page 3 is requested, so according to memory reference string, page 2 has 3rd reference and page 1 has 1st reference and page 0 has no reference ahead so it has to be replaced by page 3. Similarly other pages are replaced in the memory and the page faults are calculated.

**Total page frames = 4**

| 7 | 0 | 1 | 2 | 0 | 3 | 1 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| 7 | 7 | 7 | 7 | | 3 | | 3 | 3 | |
| | 0 | 0 | 0 | | 0 | | 0 | 2 | |
| | | 1 | 1 | | 1 | | 1 | 1 | |
| | | | 2 | | 2 | | 4 | 4 | |

Total number of page faults: 7
Success percentage = (3/10) *100 = 30%
Failure Percentage = (7/10) *100 = 70%
Success: Failure ratio = 3:7

The optimal page-replacement algorithm is challenging toimplement, because it requires future knowledge of the reference string which is practically not possible as new page requests are simultaneously generated. As a result, the optimal algorithm is used mainly for comparisonstudies.

## 3.5 LET US SUM UP

The key points we discussed are demand paging, First in First Out, Least recently used and Optimal page replacement policy and calculation of page faults, success and failure percentages and its ratios for different number of available page frames.

## 3.6 CHECK YOUR PROGRESS

### True or False

6. Main memory is also called virtual memory
7. Belady’s anomaly is generally found in FIFO
8. Valid bit is turned on by kernel if the page contents are legal
9. Optimal page-replacement algorithm is challenging to implement, because it requires future knowledge of reference string
10. The page from the main memory has to be replaced known as a victim page.

### Fill in the blanks

2. The instruction being executed, must be in _____ memory.
3. A _____ occurs when a page requested cannot be accessed asit is not in the memory
4. When a process begins execution with no pages in memory a _____ occurs for every page brought into memory
5. _____bit indicates whether the page is modified recently by a process
6. Transferring pages from main memory to secondary memory is called ____

## 3.7 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

### True or False

1. Main memory is also called virtual memory (**FALSE**)
   Virtual memory does not exist physically as main memory.
2. Belady’s anomaly is generally found in FIFO **(TRUE)**
3. Valid bit is turned on by kernel if the page contents are legal **(TRUE)**
4. Least recently used page-replacement algorithm is challenging to implement, because it requires future knowledge of reference string **(FALSE)**
   Optimal page replacement requires future knowledge of reference string
5. The page from the main memory has to be replaced known as a victim page**.(TRUE)**

**Fill in the blanks**

1. The instruction being executed, must be in _____ memory. ( **MAIN**)
2. A _____ occurs when a page requested cannot be accessed asit is not in the memory ( **PAGE FAULT**)
3. When a process begins execution with no pages in memory a _____ occurs for every page brought into memory (**PAGE FAULT)**
4. _____bit indicates whether the page is modified recently by a process ( **MODIFY**)
5. Transferring pages from main memory to secondary memory is called ____ **(SWAP-OUT)**

## 3.8 FURTHER READING

Books for further reading in above topics are:

(7) OperatingSystemsConcepts.Addision–WesleyBySilberschetzAandGalvin.

(8) OperatingSystemsPranticeHallof IndiaPvt. Ltd. ByTanenbaum.

(9) OperatingSystems.McGrawHillBookCo.ByMadnickS.&DonovanJ.J.

## 3.9ASSIGNMENTS

1. Calculate total number of page faults, success and failure percentage for memory reference string
   1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9, 5, 4, 5, 4, 2.
   Page frames = 4
   For FIFO, LRU and Optimal page replacement algorithms

2. Calculate total number of page faults, success and failure percentage for memory reference string
   1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.
   Page frames = 3
   For FIFO, LRU and Optimal page replacement algorithms

3. Calculate total number of page faults, success and failure percentage for memory reference string

7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

Page frames = 3

For FIFO, LRU and Optimal page replacement algorithms

## 3.10 ACTIVITIES

- Explain Demand Paging
- Explain Virtual memory
- Explain FIFO page replacement policy with example
- Explain LRU page replacement policy with example
- Explain optimal page replacement policy with example

## 3.11 CASE STUDIES

Compare various page replacement policies with different page frames and identify which policy is more suitable

4

# Unit 4: File Concepts

## Unit Structure

4.1. Learning Objective

4.2. Introduction

4.3. Access Methods

4.4. Directory Structure

4.5. File System Structure

4.6. Allocation methods

4.7. Free space management

4.8. Directory implementation

4.9. Redirecting input and output

4.10. Concept of Piping

4.11. Let us sum up

4.12. Check your Progress: Possible Answers

4.13. Further Reading

4.14. Assignments

4.15. Activities

4.16. Case studies

## 4.1 LEARNING OBJECTIVE

After studying this unit student will be able to understand the basic concepts of file organisation, methods by which file can be accessed, directory structures and concepts of piping and redirection

## 4.2 INTRODUCTION

This unit deals with the organisation of a file. The group of related bytes form a field, for example studentid, a group of related fields make a record, for example row of student details of studentid 1, and this records are of fixed length or variable length. Variable length records have the size according to the data entered in the record while fixed length has a static size. The group of related records make a file and group of related files interconnected at different levels make a database.

## 4.3 ACCESS METHODS

The files are group of related records and these records or information stored in the files can be accessed in one of the three ways 1) Sequential access method 2) Direct Access method 3) Indexed sequential method.

**1. Sequential Access method**

It is the simplest and most common access method of accessing the records in the file. The records are accessed in sequence that is one record after the other. The two major operations on file are read and write. The read operation read the next record and move the file pointer forward. The write operation add the records at the end of the file after adding the records. The records are accessed sequentially that is if $n^{th}$ record is to be accessed, n-1 records are skipped and then $n^{th}$ record is read. Similarly moving backward from $n^{th}$ record to $1^{st}$ record, n-1 records are skipped backwards sequentially. This method was used in accessing the records on tape drive

**2. Direct Access Method**

The records organised in the file may be fixed length or variable length. The direct access method allows the processes to read or write records in file randomly and fast. Generally direct access methods are used in databases as data can be accessed using random queries. The records in the file can be accessed in any order. The records are recognised according to their block or record number so record number is included as a parameter in read and write operations. These block numbers are relative to the starting record of the file, the record number 1 may have address 65532 and record number 2 may be at 69236. Since all the records are of fixed length it is easy to read and write the records. If we have the starting address of the file and logical record number we can easily navigate to the position. For example starting address of the file is 1000 and we have to read record number 5, and each record is of 10 bytes then the position of record would be 1000 + 10*5 = 1050.

**3.Index Sequential Method**

This method involves creation of an index for the file which is useful to search a record and directly get the pointer to the desired record. This index is just like index of any book which contains topic and its page number so we can directly go to the desired page, similarly for finding a record we have to search the index of the record. Suppose we have student data file containing studentid and marks. Studentid is of 5 digits and marks are of 5 digits, so 10 bytes are used for a student record. Suppose a block is of 1000 bytes then 100 records can be stored in a block. If there are 15000 records then there would be 150 blocks. We can define an index for first studentid in each block, so there would be 150 entries in index each of 5 bytes that is 750 bytes. We can any algorithms like binary search to search a record, but as the entries increase, index becomes large and difficult to search and manage.

## 4.4 DIRECTORY STRUCTURE

The directory can be viewed as a logical organisation of the files. The directory can be organised in several ways. The most common schemes for definingthe logical structure of a directory are

1) **Single level directory**: It is the simplest directory structure where all files are contained in the same directory. The limitation is that filenames should not

have unique names and filenames are also limited to length. When there are more number of users it complicates making the filename unique. Users may find it difficult to remember the names of all the files as the number of files increases

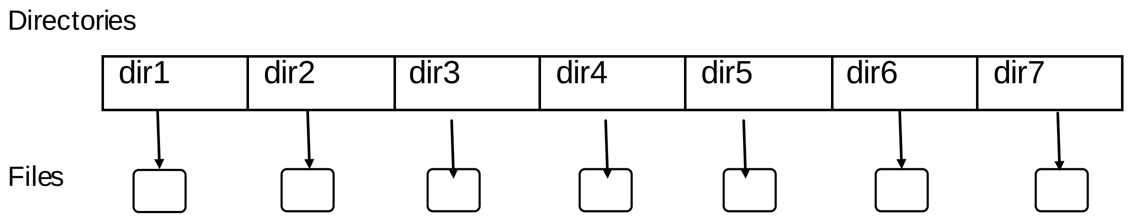


Figure 1: Single level directory

**2)Two level directory**: In single level directory the major problem is having the unique name among multiple users. The solution is to create a separate directory for each user. The two level directory structure comprises of user file directory (UFD) and master file directory (MFD). When the user logs into the system MFD is searched. It contains the user's directories, so when the user refers to a file only its UFD is searched. Every file has its own path from MFD to UFD and then to the File. The limitation is still here that the files under the UFD should have unique names, though two users having different MFD and UFD can have same name of the file in different directories. If the access is permitted by the user, then only other user can create or manage the files in other user's directories. It seems like MFD is the root directory and other directories under it are the child directories and the files act as a leaf node.

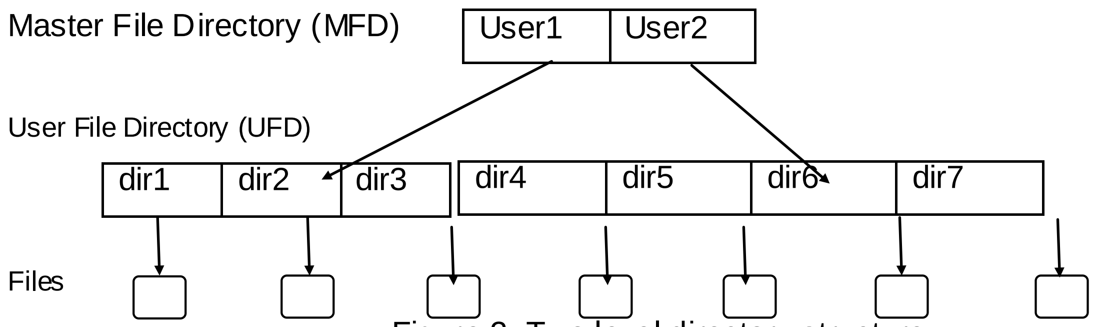


Figure 2: Two level directory structure

**3) Tree structured directory**: Extending the idea of two level directory structures to more generalised structure having different heights. User can have any number of directories at different levels and manage the files under the directories; there would be one root directory under which all other directories are created. The directories may contain sub-directories and files and to search a file user specifies the current directory in which the file is located otherwise user has to change the current directory. The system calls are provided to change the directories. The user can also provide a complete path to access the file. When the user logs in to the system he lands to a specific designated directory and then he can navigate to other sub-directories.

The path names specified by the user are of types: Absolute and Relative

Absolute path names: The path which starts from the root directory and follows the path including subdirectories up to the specific file.

For example: root/dir1/dir2/file1

Relative path names: It defines a path from the current directory. .For example dir2/file1

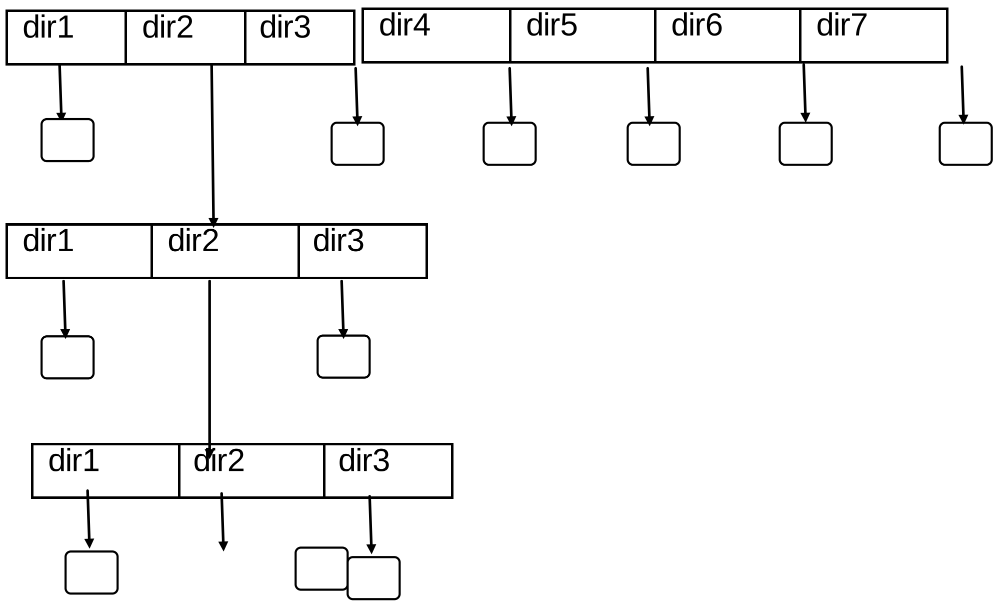


Figure 3: Tree structured directory

**4) Acyclic graph directory**: According to this structure a file or directory is commonly shared. This file or directory will exist at two places at a time. These are not the copies of same file at two places. Here only one file exists and is shared at two places so any changes made by one person are immediately reflected to the other user. If a user creates a new file it is also visible to another user. These structures are used where multiple users are working on a same project and sharing the files between the users. Acyclic graph structure is more flexible than the tree structure but the implementation is very complex as each file which is shared will have different absolute path names, that is same file is referred by different paths and other major issue is to handle the deletion of the files and directories and maintaining the consistency among the files and directories.

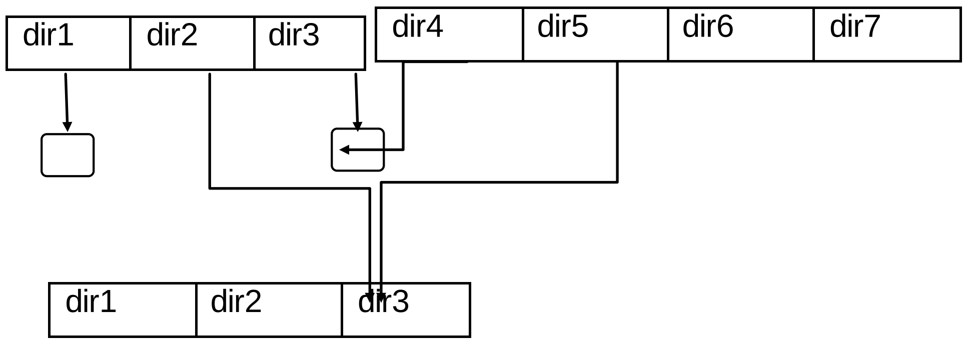


Figure 4: Acyclic graph directory

## 4.5 FILE SYSTEM STRUCTURE

The hard disks have large storage and various files are stored on this large storage. Each file has to be read or written and stored at the same place. The files need to be accessed sequentially or directly and also transfers between memory and disk areperformed in units of blocks. The file system provides an efficient way to access the disk by storing, locating and retrieving data. The main issues for designing a file system is defining a file and its properties, operations allowed on the file and how the files will be organised in directory structure. For organising directories and files, the data structures and algorithms have to create which maps the logical file system to physical storage.

The file system comprises of different levels, the lowest level is of device drivers to transfer information between disk and main memory. The device drivers convert the high level instructions into machine level instructions. Above it there are I/O controls and basic file system which issues basic commands to the device drivers to read and write to the data blocks on the disk. On top of it there is a file organisation module which keeps track of files and their logical blocks,as well as physical blocks. This module also can translate logical address to physical address. Above it there is a logical file system that manages the metadata information which includes all file system structure except the actual data of the file. File Control Block (FCB) contains information about thefile, including ownership, permissions, and location of the file contents.

Many file systems are in use today. Most operating systems supportmore than one. For example, Windows NT, 2000, and XP support diskfile-system formats of FAT, FAT32, and NTFS (or Windows NT File System). Linuxsupports over forty different file systems, the standard Linux file system isext2 and ext3. There are also distributed file systems in which a file system ona server is mounted by one or more client computers across a network.

## 4.6 ALLOCATION METHODS

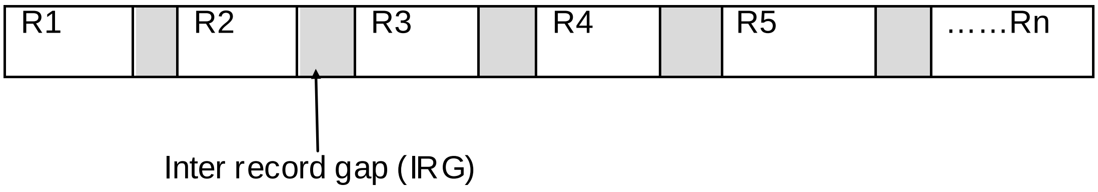
The allocation methods are used to allocate memory space to the files in such a way that the memory is utilised effectively and files can be accessed quickly. These can be done using three methods Contiguous Allocation, Linked Allocation and Index Allocation.

1. **Contiguous Allocation**

   There are two types of records namely fixed length records and variable length records. These records can be sequentially allocated to a file or they can be grouped to form a block. Therefore there are four methods by which contiguous allocation can take place

   a) Unblocked fixed length contiguous allocation

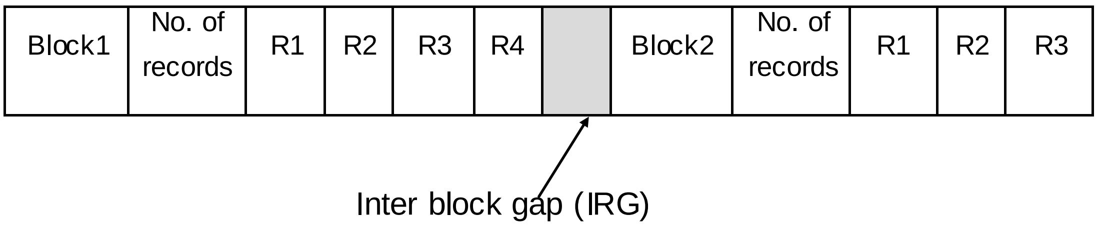
      Here the records are allocated sequentially one after the other and each record is separated by a inter record gap (IRG)



      Here the problem is if the size of the records are less than the size of inter record gap then memory used for gap is larger than the size of data stored in it. For example if the size of IRG is 5k and size of records are of 2k and if there are 100 records than 500k is used for irg and 200k is used to store records. Thus memory is not utilized effectively. The advantage of such system is we can easily navigate to the specific record. If we want to access N$^{th}$ record then we have to navigate to (N-1)*size of record in bytes.

b) Blocked fixed length contiguous allocation

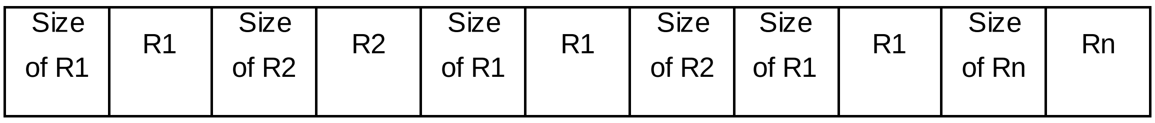
Here the records are grouped into blocks and each block is separated by inter block gap (IBG) to overcome the memory waste of IRG



The advantage of blocking is that records can be read faster and memory of IRG is not wasted.

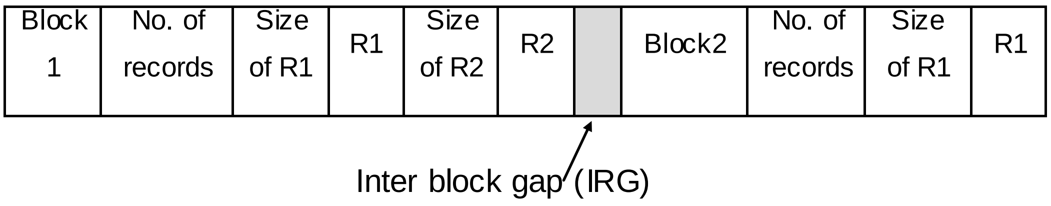
c) Unblocked variable length contiguous allocation

The other type of record is variable length record means each record will have different size so the size of the record is stored along the records and they are stored sequentially. To access the record n, then sum of size of all records before it are calculated to traverse to record n.



d) Blocked variable length contiguous allocation

Here the records are of variable length so size of record is stored before the record and the records are organised in blocks having inter block gap to separate two blocks.

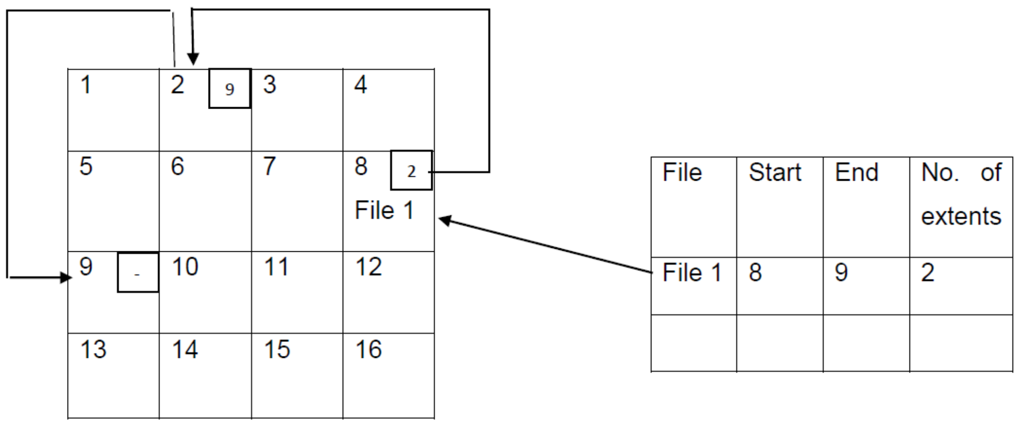


2. Non Contiguous Allocation

   The memory is allocated non-contiguously to the files that is the records are not stored in a sequence but are stored randomly at different locations. This method of allocation is carried out in two ways:

   a) Linking at the storage level

      The memory is logically divided into numbered blocks and the fragments of file are allocated to non-contiguous blocks. Initially the file is searched in the file allocation table to get the starting block. The start block contains the location of the next block and the last block contains null entry which indicates the end of the file.The disadvantage of the scheme is if the pointer to the starting block is damaged then the file would not be accessible even if present on the main memory.



| File | Start | End | No. of extents |
|---|---|---|---|
| File 1 | 8 | 9 | 2 |
| | | | |

Figure 5: Linking at the storage level

Here the example is shown for a single file file1 as the same method is applicable to multiple files in the system

   b) Linking at the directory level

      To overcome the disadvantage of linking at storage level, the file allocation table stores all the fragments of the file so if any of the link is damaged, the file could be recovered using other links of the file. The file allocation table stores the start and end of the file fragments and the last part of the file will have null entry in the end.

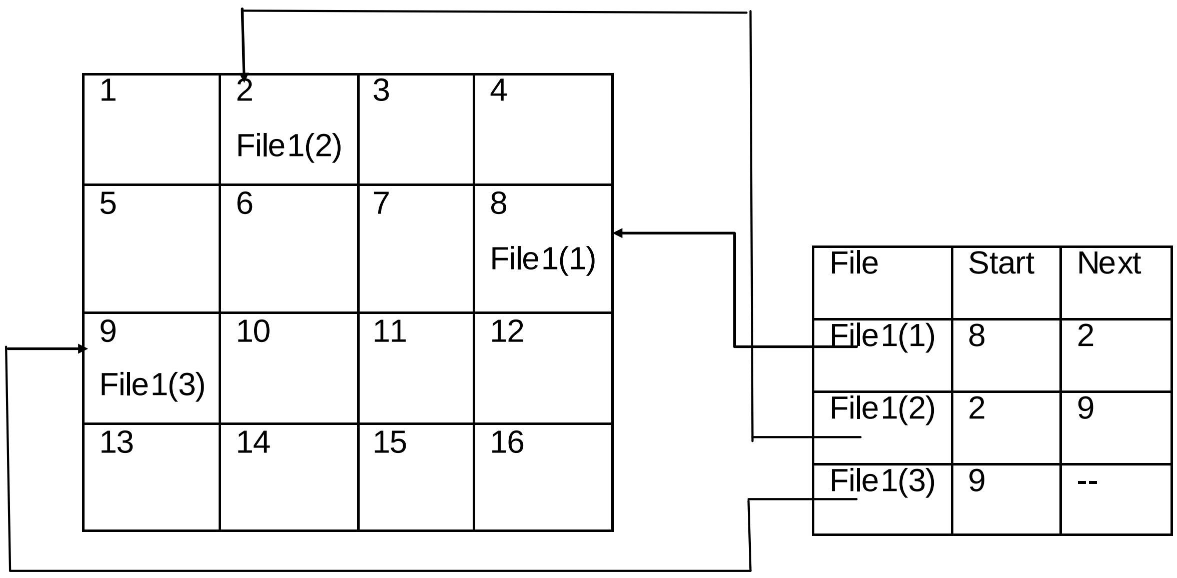


**Figure 6: Linking at the directory level**

3. Index allocation

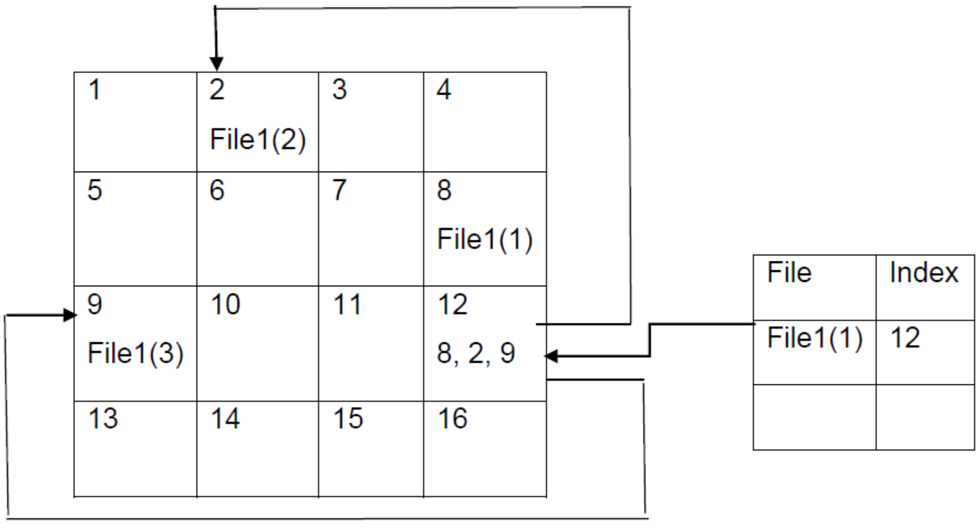


Figure 8: Index allocation

In index allocation, one block in the memory is used as an index block for a file. This block contains the pointers to the location of the file. For example file1 has index block 12 and it contains the pointers to block 8, 2 and 9. There are no more blocks which indicate the end of the file at block 9. The file allocation table contains only the index entry for the file. This method is more efficient for searching for the files.

## 4.7 FREE SPACE ALLOCATION

As disk space is limited, we have to reuse the space from deleted files for newfiles and to keep track of free diskspace, the system maintains a free list records of all freedisk blocks-those not allocated to files or directories. When a new file is created, we search the free list for the amount of space is available and then allocate thatspace to the new file. This space is then removed from the free list and whena file is deleted, its space is added to the free list. This free list can be implemented by following methods:

1) **Bit vector**

   Each block of disk space is represented by a bit. If the block is free the bit value is 0 and if the block is allocated then bit value is 1. For example for free blocks 3,6,8,11,15 and rest of the blocks are allocated, the free space bitmap will be 110110101101110.

   The main advantage of this method is its very simple to implement and efficient to find the free and occupied blocks. To find the free block, system uses a bit vector and search each bit in the vector for 0-value. The disadvantage of the method is that the whole bit vector has to be kept in the memory, so to keep track of 1GB of disk with 512 blocks, more than 300kb memory is utilised.

2) **Linked List**

   Another approach is to link the free blocks using link list and keep the link of first free block in the memory. The first block contains the pointer to the next free blocks. For example for free blocks 3,6,8,11,15 and rest of the blocks are allocated, the free list will be having the link list of 3,6,8,11,15. The pointer to block 3 will be stored in the memory which will have link to block 6, 8, 11 and 15.

3) **Grouping**

   The list approach is modified such that the list contains the free blocks in the memory and the first block is kept in the memory. Here the last block contains address of a list containing another free blocks.

## 4.8 DIRECTORY IMPLEMENTATION

The selection of directory-allocation and directory-management algorithms affects the efficiency and performance of the filesystem. The algorithms used for directory implementation are:

1) **Linear List**

   Linear list is the simplest way to implement a directory structure. It is using linear list of filenames with pointers to data blocks. Whenever a new file is created the list is searched if there is any file with same name. If there is no file with same name then an entry is made for new filename in the linear list. Similarly when a file is deleted the entry is removed from the list and free the unused space. The disadvantage of the system is entire list needs to be searched to find a directory entry which is time consuming. To avoid this frequently used directories or files can be stored in a cache and in sorted form so searching becomes easier. Alternately it can be implemented using a binary tree.

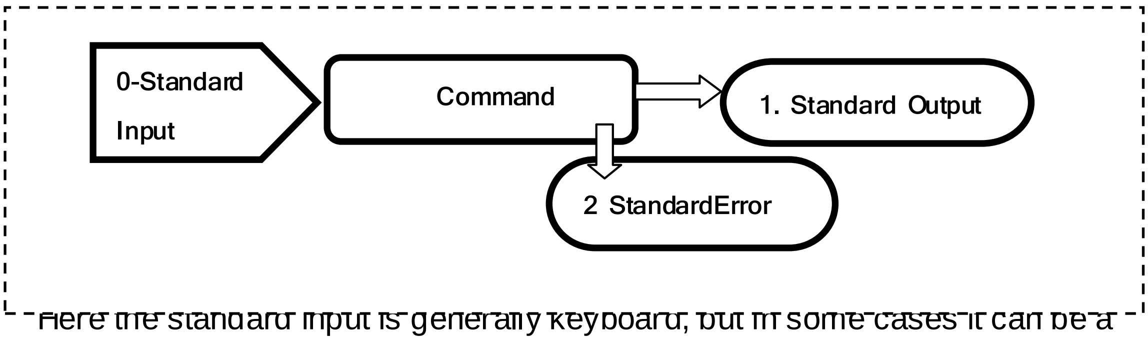
2) **Hash Table**

   The directory entries are stored in a hash table which keeps the value computed for filenames and the pointer to the filename in the linear list. This can reduce the search time. The major disadvantage of the method is that hash table has a fixed size and hash function is used for the fixed size hash table. If size of hash table is changed then hash function needs to be changed and new hash values are calculated.

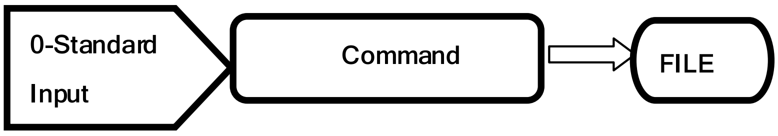
## 4.9 REDIRECTING INPUT AND OUTPUT

One of the powerful feature of Unix is input/ output redirection. The general commands of unix takes the input using input devices and displays the output on the terminal. Redirection is a feature through which we can redirect output of a command to a file, device or other commands and also give input from files to the commands. The file descriptors are assigned by operating system as 0 indicates standard input, 1 indicates standard output and 2 indicates standard error.

Each command gets input from standard input and generates output and sends it to the standard output stream and if an error is encountered then it generates standard error through error stream.



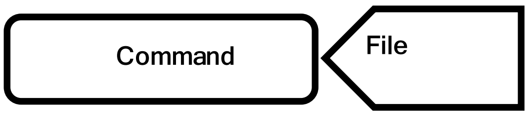
Here the standard input is generally keyboard, but in some cases it can be a redirected output of other command and the standard output can be an output file



The Linux command syntax is **command > filename**

For example:  who > list_of_users

The redirection can be used to get the input from the file and give it to the command for execution



The Linux command syntax is **command < filename**

For example:  bc< input_file

Double redirections are used to append the existing contents in the file

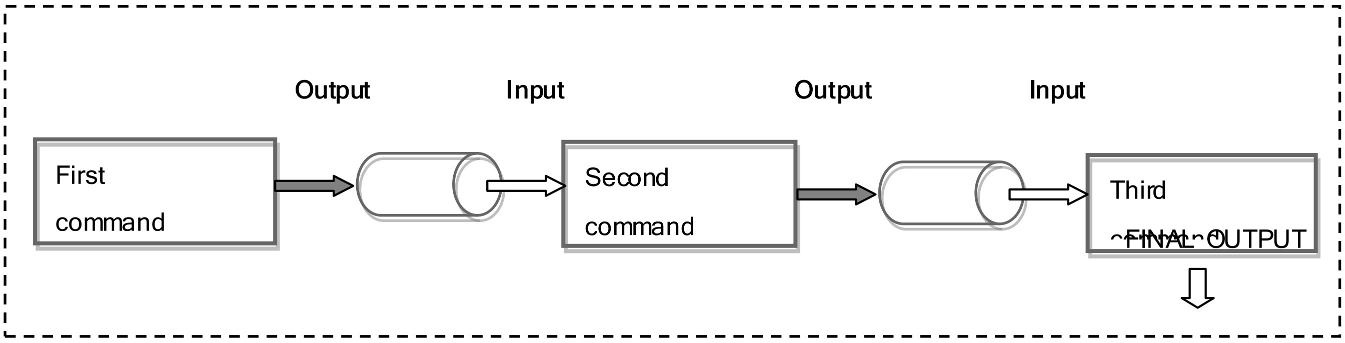
command>> filename

example : cat >> myfile

## 4.10 CONCEPT OF PIPING

Another useful feature of unix is Piping through which output of the first command can be the input of other command. The symbol used for piping is " | "

**firstcommand | secondcommand | thirdcommand**



For example:
who | wc – l
The output of who command is list of currently logged in users and this output is send as an input to wc – l command, this command counts the number of lines, so the number of lines of who command are counted by wc -l command to count number of currently logged in users.

## 4.11 LET US SUM UP

In this unit we have studied about File Access Methods, Directory Structure, File System Structure, Allocation methods, free space management, Directory implementation, Redirecting input and output and Concept of Piping

## 4.12 CHECK YOUR PROGRESS

**Fill in the blanks**

1. Records are of two types _____ and _____
2. ______ is a feature through which we can redirect output of a command to a file
3. Directories can be implemented in _____ and ______ ways
4. Grouping is a modified ________ approach.

5. In Bit vector each block is represented by a ________

**True of False**

1. Databases generally use sequential access method
2. For two level directory structure when user logs in to the system, UFD is searched first.
3. Pipes are used to transfer output of one command as input to another
4. Most popular file system of windows is ext3
5. The absolute path starts from the root directory and follows the path including subdirectories up to the specific file

## 4.13 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

**Fill in the blanks**

1. Records are of two types Fixed length and Variable length.
2. Redirection is a feature through which we can redirect output of a command to a file
3. Directories can be implemented in Linear List and Hash table ways
4. Grouping is a modified List approach.
5. In Bit vector each block is represented by a bit

**True of False**

1. Databases generally use sequential access method (FALSE)
   It uses direct access method
2. For two level directory structure when user logs in to the system, UFD is searched first. (FALSE)
   Initially MFD ( Master File Directory is searched )
3. Pipes are used to transfer output of one command as input to another (TRUE)
4. Most popular file system of windows is ext3 (FALSE)
   Ext3 is most popular file system of Linux, windows use fat32 or ntfs
5. The absolute path starts from the root directory and follows the path including subdirectories up to the specific file (TRUE)

## 4.14 FURTHER READING

Books for further reading in above topics are:

(1) OperatingSystemsConcepts.Addision–WesleyBySilberschetzAandGalvin.

(2) OperatingSystemsPranticeHallof IndiaPvt. Ltd. ByTanenbaum.

(3) OperatingSystems.McGrawHillBookCo.ByMadnickS.&DonovanJ.J.

## 4.15 ASSIGNMENTS

1. Explain Contiguous storage allocation
2. Explain non contiguous storage allocation
3. Explain piping and redirection
4. Explain directory implementation
5. Explain file system in detail

# Block-2

# Memory and File Management

# Unit 1: Basics of Linux 1

## Unit Structure

1.1. Learning Objectives

1.2. Introduction

1.3. Components of Linux System

1.4. Basic Linux Usage

1.5. Log-in and Log-out utility

1.6. Basic Linux commands

1.7. Let Us Sum Up

1.8. CheckYour Progress

1.9. Further Reading

1.10. Assignments

1.11. Activities

1.12. Case studies

## 1.1LEARNING OBJECTIVE

After studying this unit student should be able to:

- know the basic architecture of Linux OS
- use the basic Linux commands
- obtain the information about any command

## 1.2INTRODUCTION

Linux is an open source operating system developed by community for computers, embedded devices, servers and mobile devices. As it is open source, code is freely available for the use. It is distributed under open source license. Major of the computer platforms are supported by thisoperating systems. It is one of the popular version based on UNIX as it was designed considering UNIX compatibility but UNIX is a commercial operating system. Most of the UNIX and Linux commands are similar.

Linux is one of the most popular and widely used operating system or a kernel which was idea of young 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. Many of the popular operating systems like Knoppix, Ubuntu, Fedora, Debian and Slackware are powered by the Linux kernel.

## 1.3 COMPONENTS OF LINUX SYSTEM

Linux Operating System has primarily three components

- ***Kernel*** – It is the core component of the Linux. Major activities of this operating system are handled by the kernel. Low level hardware details are abstract to application or system programs by means of kernel. The kernel consists of various modules which interacts directly with the underlying hardware for providing required abstraction.
- ***System Library*** –These libraries are special routines or programs using which application programs or system utilities which uses the Kernel's

features. Most of the operating system functionalities are implemented by these libraries so do not requires kernel module's code access rights.

- ***System Utility*** –The specialized, individual level tasks are managed by the system utilities.

**Basic Features**

Following are some of the important features of Linux Operating System.

- *Portable* − Portability refers to the ability of software to works on different types of hardware. Linux kernel and application programs can be installed on the wide variety of platforms.
- *Open Source* − Linux source code is freely available and it is community based development project. Multiple teams work in collaboration to enhance the capability of Linux operating system and it is continuously evolving.
- *Multi-User* − Linux is a multiuser system means multiple users can access system resources like memory/ ram/ application programs at same time.
- *Multiprogramming* − Linux is a multiprogramming system means multiple applications can run at same time.
- *Hierarchical File System* − Linux provides a standard file structure in which system files/ user files are arranged.
- *Shell* − Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs. etc.
- *Security* − Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

**Linux System Architecture**

The architecture of a Linux System consists of the following layers–

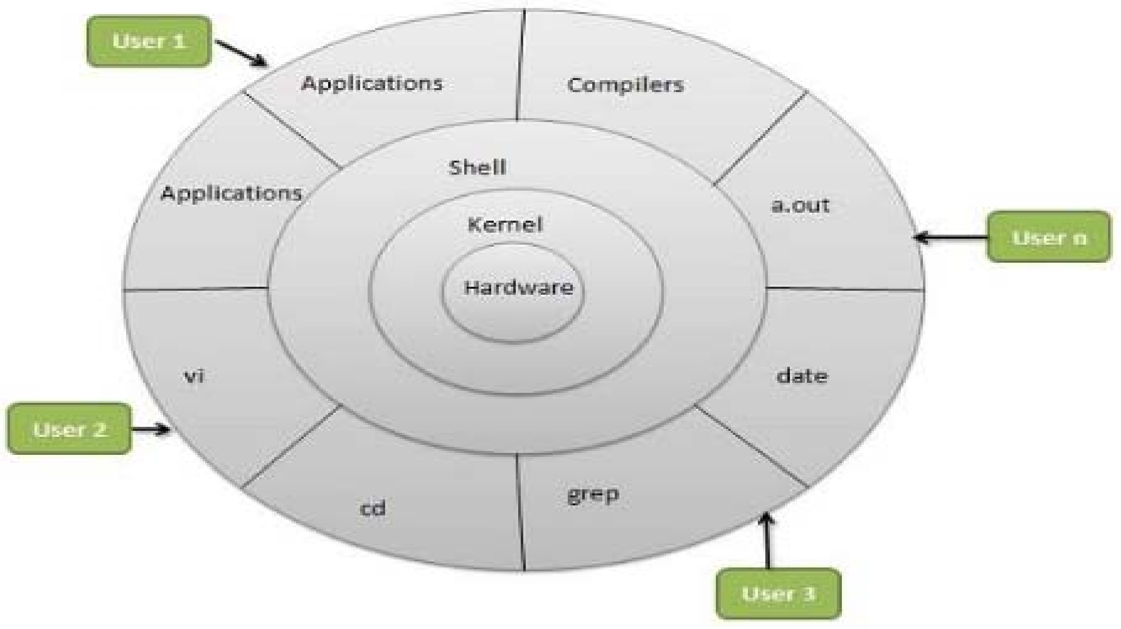


**Fig: Linux operating system**
**[Source: www.tutorialspoint.com/operating_system/os_linux.htm]**

- Hardware layer – It consists of the peripheral devices (like RAM/ HDD/ CPU etc).
- Kernel − It acts as a core component of Operating System. It is an interface between shell and low level hardware and provides abstraction by interacting directly with hardware in order to provide low level services to upper layer components.
- Shell – It acts as an interface to the kernel so as to hide the kernel's functionalities from the users. It accepts commands from the user and executes corresponding kernel's functions.
- Utilities – It comprises of the various functionalities of an operating systems.

## 1.4 BASIC LINUX USAGE

Linux is known as Multi-tasking, Multi-user, and Multi-programming Operating system.So, multiple users are login into the terminal simultaneously, each running many programs.

A command basically comprises of the set of instruction provided by a user instructing the computer to do some task. It can be a single program or a group of linked programs. A command is generally issued to the shell by typing it at the command line and then pressing the ENTER key.

A shell is itself a program which interprets the commands issued at the command prompt and then executes (i.e., runs) them. Shells is an interface between the user and the kernel which in turn helps the user to interact with the system. The default shell on most Linux systems is bash but most have several.

## 1.5LOG IN ACTIVITY, LOG OUT ACTIVITY

Since the UNIX and Linux are the multi-user operating system, the user are provided with their own credentials by the system administrator. Each user has a personal environment (shell, home directory, profile etc.)

Basically, two types of user are there:

1. Normal users having the restricted rights
2. System administrator (also known as root) with all privileges.

The later is responsible for the installation, configuration and maintenance of the system as well as the user administration.

Every user has their own credentials for logging into the system. Each user account is protected by a password.

**Login command**

Login is a command to begin a new session with the system. It is normally invoked automatically by responding to the login: prompt on the user' terminal.

At the login prompt type your account name (in lower case letters) followed by ENTER or RETURN.

```
Login: student
Password:
```

Where student is a user name (also known as a userid or login name). The string for password that you entered will not be displayed on the screen. If you enter the login name or password incorrectly, the system will display the following message.

```
Login incorrect
Wait for login retry
```

After successful login system will show $ as a prompt. If you are the system administrator and logged in as root. It will give the prompt#

In Linux the prompt will be

```
[root@localhost]#
```

### Logout command

Logout is a command used by normal users to end their own session. If the user has left the terminal logged in and some other unauthorized users can use this terminal and access the confidential data.

To logout from current user session:

```
$ logout
```

*output:*

no output on screen, current user session will be logged out.

## 1.6BASIC LINUX COMMANDS

### Your First Keystrokes

Let's get started and launch a terminal emulator. Once invoked, you would see something like this:

```
[user@localhost ~]$
```

This is the shell prompt which is ready to accept the input from the user.

The terminal in Linux allows to run the Linux commands which is just like a windows operating system command prompt. All the commands in Linux are case-sensitive. The terminal is a user interactive and execution of the command is done after the pressing of the enter key.

A name of the command basically a set of one or more strings that comprises of arguments and options separated by the white spaces or tabs.

The general syntax for the commands is

```
command [-options] [-arguments]
```

The square brackets indicate that the enclosed items are optional. Most commands have at least a few options and can accept (or require) arguments. However some commands do not accept any arguments.

Let us discuss the various general purpose Linux commands.

1. **pwd**: It stands for print working directory. When you open the terminal, it shows the directory in which user is currently working. It gives us the path that starts from the root i.e., absolute path. The base of the Linux file system is the root which is denoted by a forward slash ( / ). The user directory is something like “/home/username”.

   Syntax: $pwd

   ```
   root@tryit-just:~# pwd
   /root.
   ```

   It indicates that user is in root directory. Our current working directory is set to our home directory when we log in to the system.

   ```
   [user@localhost ~]$ pwd
   /home/user
   ```

   Each user account is given its own home directory, the place where user is allowed to write the files.

2. **date**: the date commands tells us the current time and date. Also, it allows us to set date and time. The default output of the date command displays the

date in the time zone on which linux operating system is configured. You must be the root user (Super user) to change the date and time.

```
Syntax: $date [OPTION]... [+FORMAT]
```

The date command can also be used with the following format specification.

| Format | Purpose |
|---|---|
| $date +%m | Month |
| $date +%h | Month name |
| $date +%y | Last two digits of a year |
| $date +%H | Hours |
| $date +%M | Minutes |
| $date +%S | Seconds |
| $date + %D | Display date |
| $date + %Y | Display the last two digit of the year |

Syntax:

$date +% [format-option]

Examples:

```
$date "+%D"
```

**Output:**

```
07/04/19
```

**Command:**

```
$date "+%D %T"
```

**Output:**

```
07/04/19 15:03:24
```

**Command:**

```
$date "+%Y-%m-%d"
```

3. **who:**It provides the list of the users currently logged into the system. It returns the username, the terminal and login time of the user. The terminals are actually the device file stored in /dev directory. This command is generally used to monitor that terminals are utilized efficiently and identify the activities of the users and their idle time.

   ```
   $who
   $who –Hu
   ```

   -H is for header display for each column, by default the header information is not displayed and –u option displays the idle time for each user and the process id. If the user is inactive for more than 24 hours, then it is displayed as old

   The idle column shows that student seems to be idling for last 30 minutes. PID shows the process identification number.

4. **Who am i:**The who command with the arguments "am" and "I", displays a single line output only which includes the login detail of the user who invoked this command.

   ```
   $ who am i
   Student tty01 Apr 07 03:17
   ```

5. **Passwd command:**This command is used to change the password. It first asks for the old password which is checked with the stored password for this user. It will ask for the new password and third time for confirming the new password if it is a valid. The normal user can change their password only while super user can change the password for any account.

   ```
   [user@localhost ~]$ passwd.
   (current) UNIX password:
   New UNIX password:
   ```

This command also enforces for the use of “strong” passwords. That is it will refuse to accept passwords that are dictionary words, too similar to previous passwords, too short, or too easily guessed:

```
[user@localhost ~]$ passwd.
(current) UNIX password:
New UNIX password:

BAD PASSWORD: is too similar to old one

New UNIX password:

BAD PASSWORD: is based on dictionary word
```

User with super-user privileges can specify a user name as an argument to the passwd command to set the password for another user. Super user also can lock the account and set the expiration, etc.

6. **Man command:**The man command in is used to display the user manual of any command which is supplied as an argument to the man command. It display a detailed view of the command which information like NAME, SYNOPSIS, DESCRIPTION, OPTIONS, EXIT STATUS, RETURN VALUES, ERRORS, FILES, VERSIONS, EXAMPLES, and AUTHORS.

```
$man [OPTION]... [COMMAND NAME]...
```

**1. No Option**: It displays the whole manual of the command.

**Syntax:**

```
$ man [COMMAND NAME]
```

**Example:**

```
$ man printf
```

**2. Section-num**: Since a manual is divided into multiple sections so this option is used to display only a specific section of a manual.

**Syntax :s**

```
$ man [SECTION-NUM] [COMMAND NAME]
```

**Example:**

```
$ man 2 intro
```

**3. -f option**: One may not be able to remember the sections in which a command is present. So this option gives the section in which the given command is present.

**Syntax:**

```
$ man -f [COMMAND NAME]
```

**Example:**

```
$ man -f ls
```

## 1.7 LET US SUM UP

**Points to ponder**

- Linux is a multi-user and multi-tasking operating system.
- Shell work as an interpreter between the user and the kernel
- Kernel is the core part of an operating system
- All the commands in Linux are case-sensitive
- Date command is used to display date and time
- Man is a command for getting the help of other commands
- Passwd command enforce to make use of strong passwords.
- Super user can specify a user name as an argument to the passwd command to set the password for another user.

## 1.8 Check your progress

**State true or false**

1) The user can directly communicate with the shell.
2) Linux is a proprietary operating system.
3) Shell is a core component of an operating system.
4) The shell acts as an interface between the user and the kernel.
5) Login command is used to begin a new session with the system.
6) Passwd command does not enforce to use the strong password

**Match the following**

| | |
|---|---|
| 1) Who | a) help manual |
| 2) Man | b) print current working directory |
| 3) Pwd | c) list the current logged user |
| 4) Passwd | d) changing the password |
| 5) Date | e) display the time |

**Multiple choice questions**

1. Which command will display the help pages for any given command in Linux?

   a) man  b) help  c) help pages  d) all the above

2. What do you mean by a shell?

   a) is a program that is used for giving inputs to the machine.

   b) is a third party software for doing programs.

   c) is a virus

   d) None of the above

3. ________command is used for finding current directory?

   a) pwd

   b) cwd

   c) ewd

   d) none of the above

4. --------- command can be used for changing the password

   a) passwd

   b) password

   c) setpasswd

   d) both a & c

5. ------ is the home directory of the user root

   a) /

   b) /root

   c) /home/root

   d) /etc/root

6. Which one of the following is not applicable to Linux operating system?

a) portable  b) open source  c) multitasking  d) single-user OS

## 1.9 FURTHER READING

1. Here is an article about the concept of shells in computing: http://en.wikipedia.org/wiki/Shell_(computing)

2. https://www.geeksforgeeks.org/interesting-facts-about-linux/

## 1.10 ASSIGNMENTS

1) Discuss in detail, how the man command is useful for learning the other Linux commands?
2) Explain the use of the pwd command in Linux
3) Write a short note on: The role of the System Administrator in Linux
4) Install any UNIX based operating system like FEDORA, UBUNTU in your computer.

## 1.11 ACTIVITIES

**Q1. Execute the following commands:**

1) Display the date and time after 3 days
2) Display the date and time of previous day
3) Display the full week days
4) Display the abbreviated name for week days
5) Display the minute and second
6) Display the help of date command
7) Print the current working directory

Q2. All the student login in to the Linux system by their respective user id and password. You are supposed to display the information of all active users.

## 1.12 CASE STUDIES

1) Prepare a comparative chart of various UNIX like operating system.

# Unit 2: File System and Related Commands

2

## Unit Structure

2.1. Learning Objectives

2.2. Introduction

2.3. File Related Commands

2.4. Directory Related Commands

2.5. User and Group Permission

2.6. Let Us Sum Up

2.7. CheckYour Progress

2.8. Further Reading

2.9. Assignments

2.10. Activities

2.11. Case Studies

## 2.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- create or modify the files and directories
- use the general purpose commands on Linux file system
- set the permission of file and directories
- create user and group

## 2.2 INTRODUCTION

This unit discusses the types of files in UNIX based Linux operating system. Linux files and directory related commands makes it a popular, powerful and secure operating system. All the devices like printer, keyboard, printer, etc. are treated as files. Various commands are explained in order to provide the basic understanding of accessing the files in Linux, organizing the files and assigning the permissions etc. A file system is a logical collection of files on a partition or disk. A partition is a container for information and can span an entire hard drive if desired.

**Types of Files**

There are three types of files: **1)** Ordinary files and **2)** Directory files **3)** Device files

**Ordinary File**

The common type of files in Linux are ordinary files which store the data in a specific format on a secondary device. Ordinary file can be an executable file, program code, data file or external UNIX commands. These are also known as regular files and store the data either in binary or text format.

**Directory File**

A directory is a special file whose sole job is to store the name of the file names and their related information. All the special, ordinary or directory files are stored in the directories. The two files with the same name cannot be there in the directory. Every files has its own unique Id. In Linux, there is well organized directory structure. Linux/UNIX directory structure include various directories like root(/), etc, mnt, var, bin, dev etc., where each directory contains specific files.

**Device File**

Linux/UNIX treats all the physical devices as files only. The devices may include printer, terminals, secondary storage like USB, hard disk, CD-ROM etc. There are no special commands for handling these files. Ordinary command with their respective options are used on this device files. Most of the devices files are found in /dev directory.

In general, everything is a file and following file system objects can be found

- ‘normal’ (text-) files
- executable files (binary files or shell scripts)
- directories
- device files
- pipes
- symbolic or hard links (references to files)

All files and file system objects are ordered within a hierarchical file tree with exactly one root directory ‘/’.

Here are some common and useful bash commands, most of which deal with file and directory manipulation:

## 2.3 FILE RELATED COMMANDS

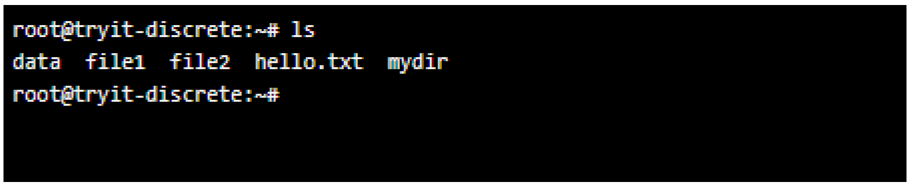
1. **ls (list directory) :**

   It lists the files and directories in the current directory. The files and directories are organized as per ASCII sequence having the number first, then the upper case and followed by lower case filenames.

**Syntax :**

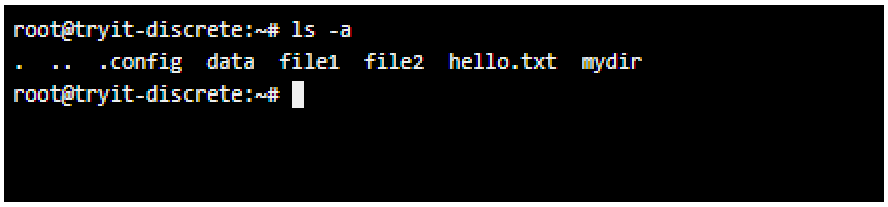
```
ls [options] [filename]

ls [options] [filename] [wildcards]
```

**Example :**

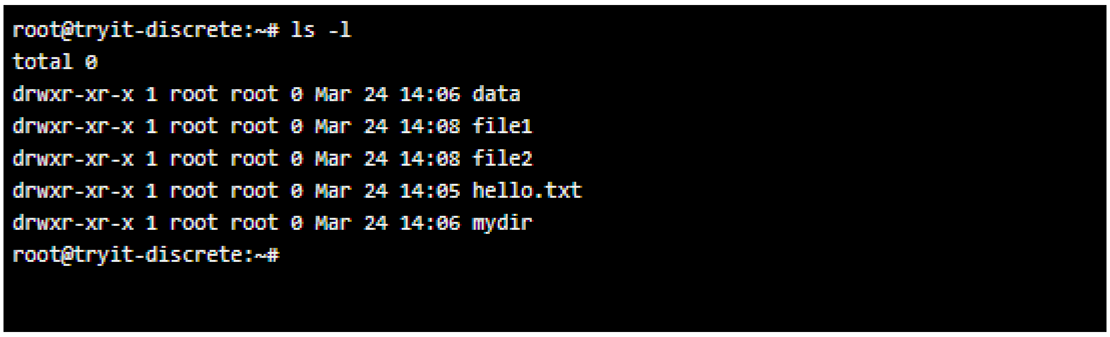


```
root@tryit-discrete:~# ls
data  file1  file2  hello.txt  mydir
root@tryit-discrete:~#
```

-a  is option which  list out all files/directories with hidden ones ( begin with a dot).



```
root@tryit-discrete:~# ls -a
.  ..  .config  data  file1  file2  hello.txt  mydir
root@tryit-discrete:~#
```



```
root@tryit-discrete:~# ls -l
total 0
drwxr-xr-x 1 root root 0 Mar 24 14:06 data
drwxr-xr-x 1 root root 0 Mar 24 14:08 file1
drwxr-xr-x 1 root root 0 Mar 24 14:08 file2
drwxr-xr-x 1 root root 0 Mar 24 14:05 hello.txt
drwxr-xr-x 1 root root 0 Mar 24 14:06 mydir
root@tryit-discrete:~#
```

-l is option gives long listing format. It displays the seven attributes which includes permissions, links, user and group, file size, last modified time stamp, and file name.

The first letter in the first column of the output window specifies the permissions and type of the file. The file type can be ordinary file ( _ ), block special file (b), character special file (c) or directory (d).

Next, it displays the file permission which include read (r), write (w), and execute (x) permissions for user, group and others.

**User**:  rwx
**Group**: rwx
**Other**: rwx

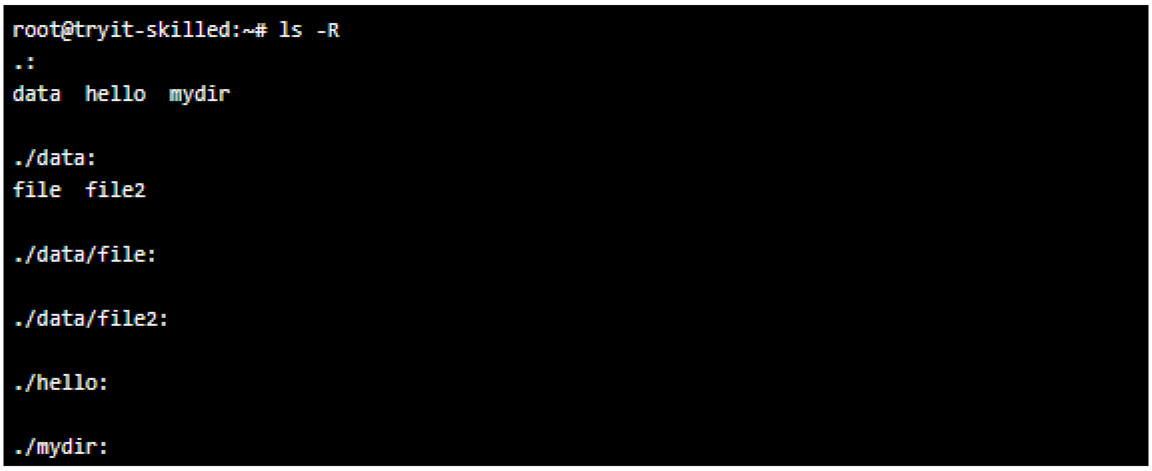
Links indicates the number of files linked to this directory or file.

Owner means the name of the user who owns or created this directory or file and if the owner is a part of the group then group name is also displayed.

The file size displays the number of bytes occupied by the file or directory.
The last modified displays the last date and time of modification and last column display the file name.

ls -d : displays the name of the working directory
ls -nd: displays the user group id along with long list format
ls -ld: displays the working directory along with long list format.
ls -r: sorts the files in reverse order
ls -t: sorts the files by modification time
ls -u: sorts the files by access time

R is option used for list directories, all sub-directories will be displayed recursively.(means display the sub files/directories within the main directory).



```
root@tryit-skilled:~# ls -R
.:
data  hello  mydir

./data:
file  file2

./data/file:

./data/file2:

./hello:

./mydir:
```

Wildcards (meta characters)

Meta characters have a special meaning in Unix. For example, ***** and ? are meta characters. We use ***** to match 0 or more characters, a question mark (**?**) matches with a single character.

For example –

$ls temp*.txt

It displays all the file names starting with temp and ending with **.txt**

$ls *.txt

Here * works as a meta character which matches with any character but ending with **.txt.**

**2. cat**

Cat command is used for creating the new files and displaying the data of already created file on the terminal. It also accepts the multiple file names as arguments.

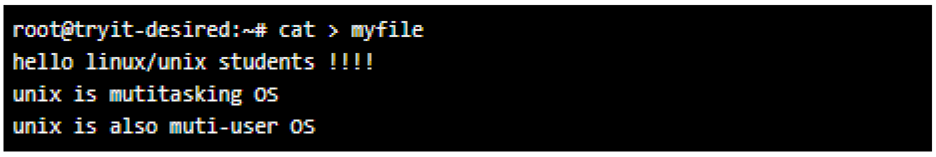
**For creating a file:-**

cat> filename

Hello welcome to my first file

[Ctrl +d]

It will result into creation of a new file with the above mentioned content.



```
root@tryit-desired:~# cat > myfile
hello linux/unix students !!!!
unix is mutitasking OS
unix is also muti-user OS
```

**To display content of file:-**

cat filename

It will display the content of the file filename on the terminal.

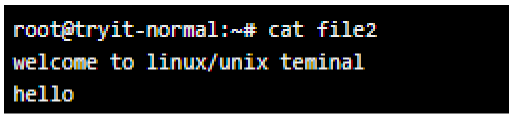
**To display the content of more than file:-**

cat filename1 filename2

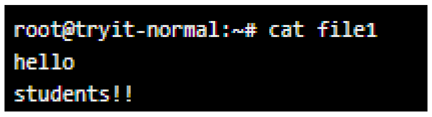
This is filename1

This is filename2

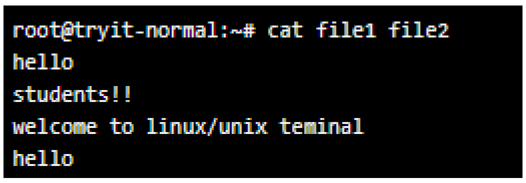
Contents of the second file are shown immediately after the first without any header information. In this way, cat can concatenate two files.



```
root@tryit-normal:~# cat file2
welcome to linux/unix teminal
hello
```



```
root@tryit-normal:~# cat file1
hello
students!!
```

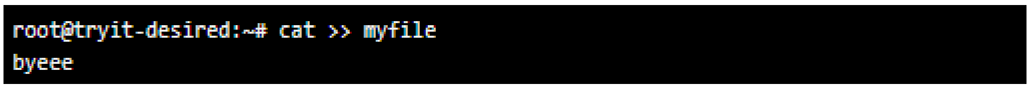


```
root@tryit-normal:~# cat file1 file2
hello
students!!
welcome to linux/unix teminal
hello
```

cat command is generally used for displaying the text files only. The executable file when displayed with cat, will show the junk.

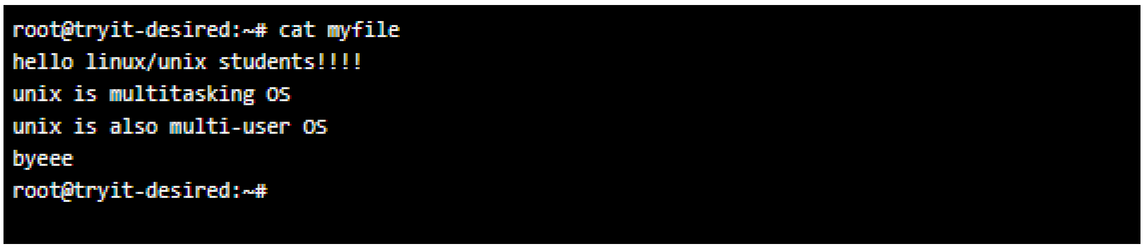
For appending the data file in existing file:-

Cat >> filename



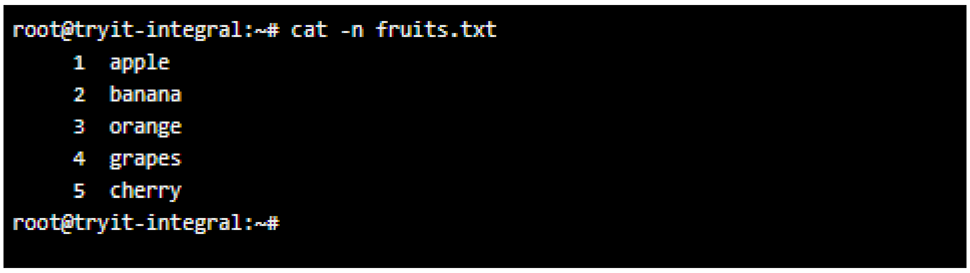
```
root@tryit-desired:~# cat >> myfile
byeee
```

After appending the data in myfile:



```
root@tryit-desired:~# cat myfile
hello linux/unix students!!!!
unix is multitasking OS
unix is also multi-user OS
byeee
root@tryit-desired:~#
```

**To display Number of lines in a file (-n) option is used.**



```
root@tryit-integral:~# cat -n fruits.txt
     1  apple
     2  banana
     3  orange
     4  grapes
     5  cherry
root@tryit-integral:~#
```

## 3. cp (Copy command)

Copy command copies a file or group of files. It creates an exact image of the file or directory on the disk with different name. It syntax needs two files to be specified in the command line. The first file name is the source file and second one is the destination file name i.e., second will be the exact clone of the first one.

The source file must be the already existing file. If the source file does not exists then it will display the error message "No such file or directory". If the destination file i.e., file2 does not exist, it will be first created with the specified name having the same content as the source file. If the destination path is not specified then current directory is treated as the default path. The source file and destination files in the same directory cannot have the same name.

**Syntax :**

**cp [options] sourcefile destinationfile**

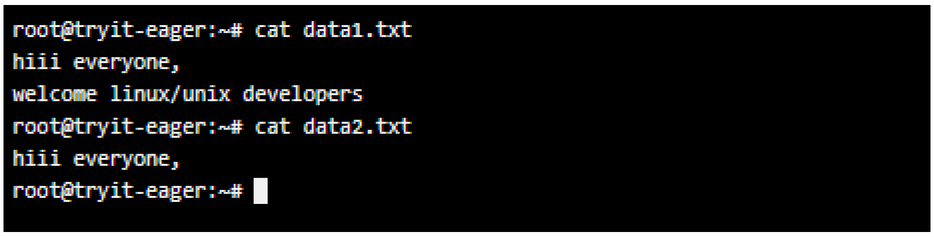
**eg. Cp file1 file2**

**cp [options] sourcefile(s) directory name**

**cp f1 f2 f3 Directory1**

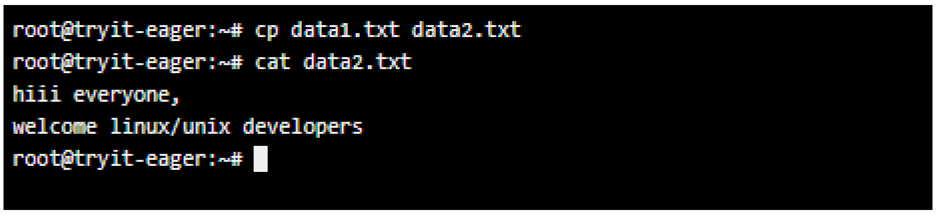
**Example :**

Before executing the cp command, let's create two files data1.txt and data2.txt with different content as below shown:



```
root@tryit-eager:~# cat data1.txt
hiii everyone,
welcome linux/unix developers
root@tryit-eager:~# cat data2.txt
hiii everyone,
root@tryit-eager:~#
```

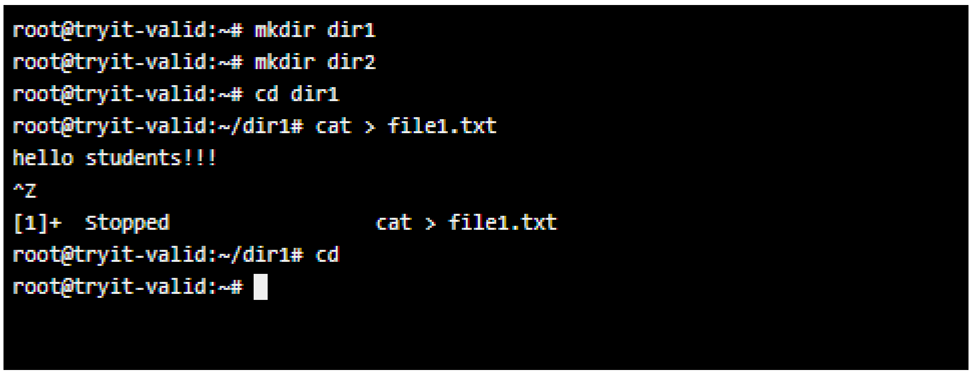
Now copying the contents of data1.txt to data2.txt. In this, the content of data1.txt are overwritten on data2.txt.



```
root@tryit-eager:~# cp data1.txt data2.txt
root@tryit-eager:~# cat data2.txt
hiii everyone,
welcome linux/unix developers
root@tryit-eager:~#
```

.

Now, let us create two directories named dir1 and dir2. Also creating one file in dir1 as show in the figure.

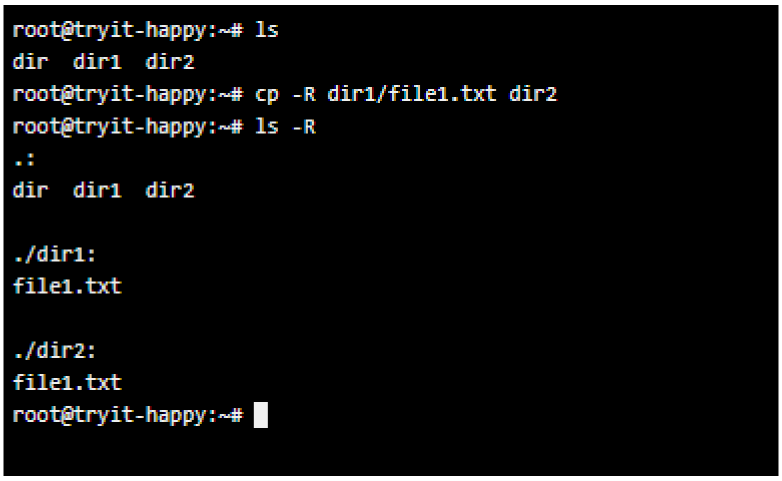


```
root@tryit-valid:~# mkdir dir1
root@tryit-valid:~# mkdir dir2
root@tryit-valid:~# cd dir1
root@tryit-valid:~/dir1# cat > file1.txt
hello students!!!
^Z
[1]+  Stopped                 cat > file1.txt
root@tryit-valid:~/dir1# cd
root@tryit-valid:~#
```

We will try to copy file from one directory to another directory so

i) Create two directory (eg: - dir1, dir2) as shown above also create one sub-file inside first directory (e.g.:- dir1/file1.txt).s

ii) Apply the cp command to copy the file from dir1 to dir2.



```
root@tryit-happy:~# ls
dir  dir1  dir2
root@tryit-happy:~# cp -R dir1/file1.txt dir2
root@tryit-happy:~# ls -R
.:
dir  dir1  dir2

./dir1:
file1.txt

./dir2:
file1.txt
root@tryit-happy:~#
```

We can't directly use cp command along with use -R option to copy as shown above.

**Options**

**cp –i: It refers to the interactive copy**

By passing this option on the command line, it will ask the user whether the file should be overwritten or not in case if the destination file already exists. If yes is specified then file will be overwritten else cp command will be terminated

$ cp –i f1.txt f2.txt

cp: overwrite ‘f2.txt’ ? y

cp –R: it refers to recursive copy.

It allows to copy directories and all its sub directories/files contained in it to a newly specified directory. If the destination directory does not exist then first the directories are created and then the files are copied to them

```
cp -R source_directory destination_directory
```

```
$ ls dir1/

f1.txt f2.txt f3.txt~ dir2 dir3
```

It will display the following error if –R is not used.

```
$ cp dir1 dir4

cp: -r not specified; omitting directory dir1
```

With -R, execute successfully

```
$ cp -r dir1 dir4

$ ls dir4/

f1.txt f2.txt f3.txt~ dir2 dir3
```

**Copying using * wildcard:**

The * wildcard to any all files or directories. It consumes lots of time when we have to copy many files from one directory to another one by one. This task can be made simpler using * wild card.

Initially assume directory dir1 is empty

```
$ls f1.txt f2.txt f3.txt dir1

$cp *.txt dir1

$ls dir1

f1.txt f2.txt f3.txt
```

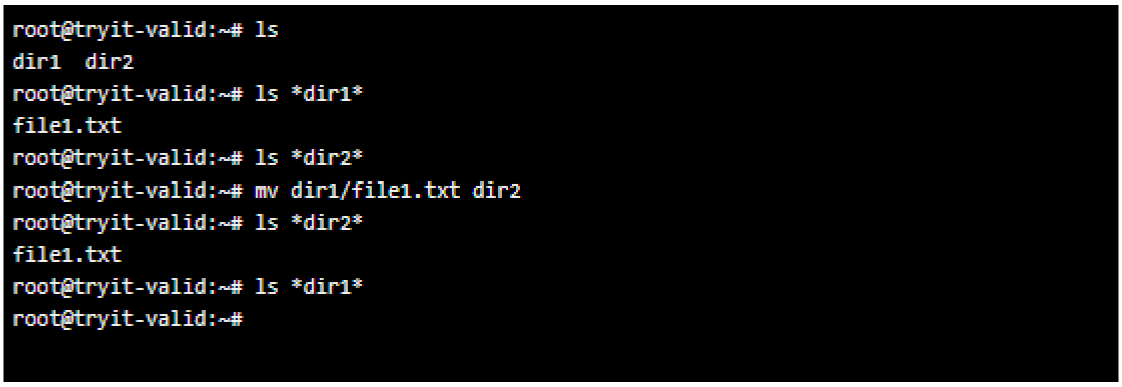
## 4. Mv (Move Command)

mv is used to move one or more files or directories from one place to another in file system. It can rename a file or directory or move a group of file to a different directory. mv command does not create a copy of the file.

**Syntax :**

**mv [source] <file name> [destination] <new/exist_file name>**

**Example :**



```
root@tryit-valid:~# ls
dir1  dir2
root@tryit-valid:~# ls *dir1*
file1.txt
root@tryit-valid:~# ls *dir2*
root@tryit-valid:~# mv dir1/file1.txt dir2
root@tryit-valid:~# ls *dir2*
file1.txt
root@tryit-valid:~# ls *dir1*
root@tryit-valid:~#
```

Now, when we want to cut one file from one location and paste to another location then mv command is used as above. If the destination file does not exist, it will be created.

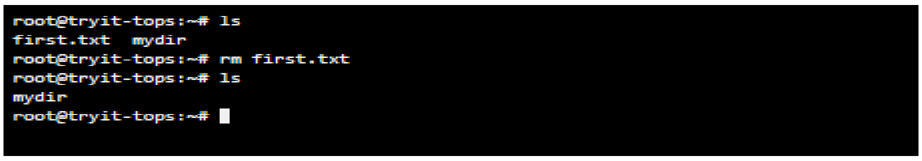
## 5. rm (remove files or directories)

rm command is used for removing/deleting files only. In actual, only link to the file is deleted which means file is not physically deleted from the hard dist. If all the other links are deleted then the file is said to be removed. The system will ask for confirmation before deletion and is deleted only if the user has the permission to do the same.

**Syntax :**

**rm [options] < filename>**

**Example :**



```
root@tryit-tops:~# ls
first.txt  mydir
root@tryit-tops:~# rm first.txt
root@tryit-tops:~# ls
mydir
root@tryit-tops:~# █
```

**Options:**

**rm – R: Recursive removal**

It will remove the directory, its sub directories and files contained in it. First of all, the files are removed and finally directories are removed.

**rm –f: remove forcibly**

File will not be removed if it is write protected. Force (-f) overrides the protection and removes the files even if they are write protected.

**Using wild card**

**rm –f  f*.txt**

wild characters will remove the file(s) according to the specified pattern. Here all the file names starting with the character **f** and with extension **.txt** would be forcefully removed.

## 2.4 Directories Related Commands

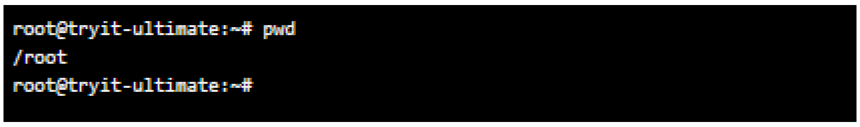
### 1. pwd (print working directory):

pwd command is use to print the name of current  working directory of the user.  As an output, it will show the absolute path of the current directory. It contains only two command flags as an options.

**Syntax :**

```
pwd [options]
```

**Example :**



```
root@tryit-ultimate:~# pwd
/root
root@tryit-ultimate:~#
```

**[options]**

**pwd -L prints the symbolic path**

**pwd -P prints the actual path which is same as pwd without any option.**

```
user@localhost:~/logs$  pwd
/home/user/logs
user@localhost:~/logs$  pwd -L
/home/user/logs
user@localhost:~/logs$  pwd  -P
/var/log
```

In the above given example the directory /home/user/logs/ is a symbolic link for a target directory /var/logs/

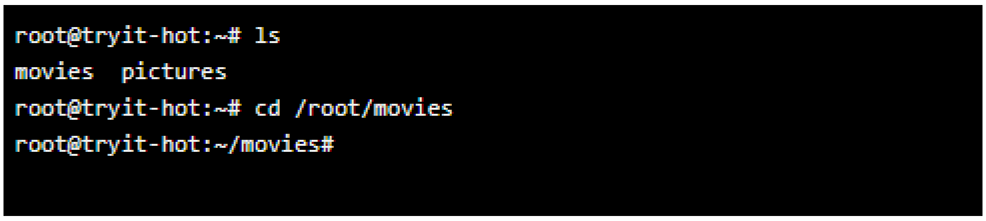
## 2. cd (changing the directory):

cd command is use to move from the current working directory to other specified directory. Now after changing the directory, the current directory is changed to the specified directory. The path names specified by the user can be relative or absolute.

**Syntax :**

**cd<directory>**

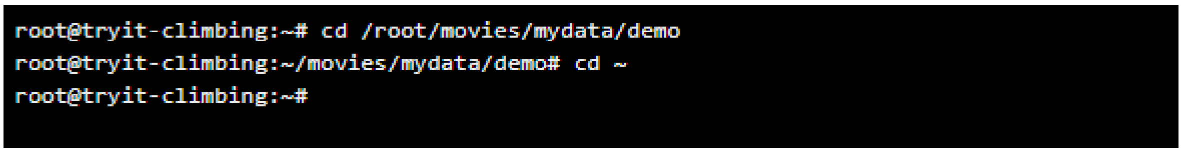
**Example :**



```
root@tryit-hot:~# ls
movies  pictures
root@tryit-hot:~# cd /root/movies
root@tryit-hot:~/movies#
```

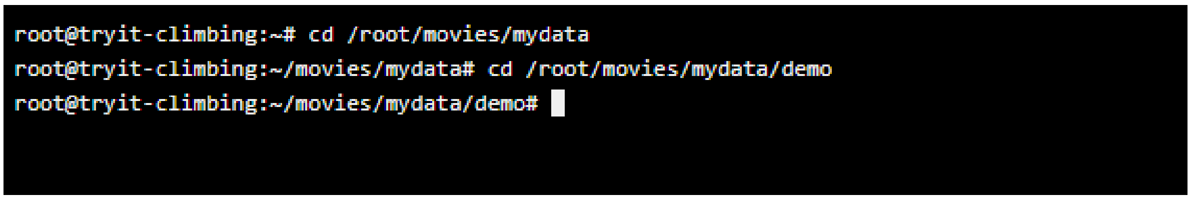
In, above example cd command is use to switch to another directory.

**Brings you to your home directory.**



```
root@tryit-climbing:~# cd /root/movies/mydata/demo
root@tryit-climbing:~/movies/mydata/demo# cd ~
root@tryit-climbing:~#
```

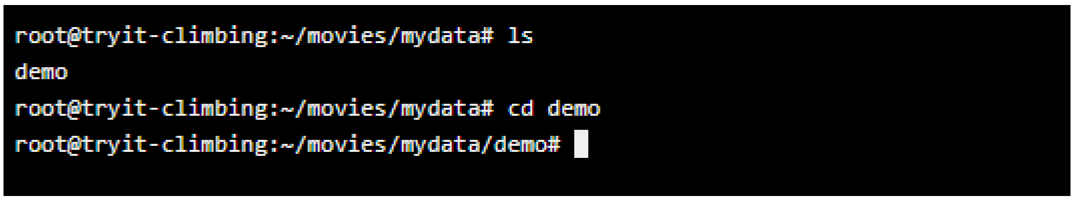
**Change directory using absolute path**



```
root@tryit-climbing:~# cd /root/movies/mydata
root@tryit-climbing:~/movies/mydata# cd /root/movies/mydata/demo
root@tryit-climbing:~/movies/mydata/demo# █
```

In above example, we want to change our directory to 'demo' from 'mydata'. So we are providing the whole path /root/movies/mydata/demo starting from the root (/). This is called absolute path.

**Change directory using relative path**



```
root@tryit-climbing:~/movies/mydata# ls
demo
root@tryit-climbing:~/movies/mydata# cd demo
root@tryit-climbing:~/movies/mydata/demo# █
```

In above example, we want to change our directory to 'demo' from 'mydata'. So we don't need to change the whole path /root/movies/mydata/demo starting from the root (/). This is called relative path.

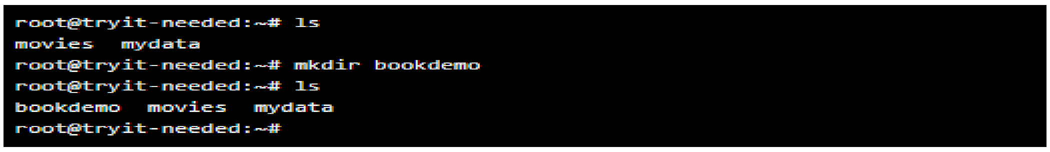
**3. mkdir (make directory):**

mkdir command is used to create an empty directory or new directory. The command is followed by the names of the directory to be created. The directories are also referred as a folders in some operating systems. This command can be used to create multiple directories at once if the user have the enough permissions. It will also allow to set the permission for the directories.

**Syntax:**

**mkdir [options] [directory]**

**Example:**



```
root@tryit-needed:~# ls
movies   mydata
root@tryit-needed:~# mkdir bookdemo
root@tryit-needed:~# ls
bookdemo   movies   mydata
root@tryit-needed:~#
```

**mkdir - -help:** It will show the help information regarding mkdir command

**mkdir - -v:** v stands for verbose i.e., for every directory created, message will be displayed.

```
user@localhost:~# mkdir –v D1 D2
mkdir: created directory ‘D1’
mkdir: created directory ‘D2’
```

**mkdir –m: allows to set the permission for the created directories**

**syntax: mkdir -m a=rwx [directories]**

```
user@localhost:~# mkdir –m a=rwx D1
```

The above command inform to the shell for setting the read, write and execute permission for all the users on the created directory D1.

Sometimes the system refuses to create a directory due to the below mentioned reasons:

- The directory with that name already exists.
- There may be an ordinary file by that name in the current directory.
- The user does not have the permission to create a directory.

## 4. rmdir (remove directory):

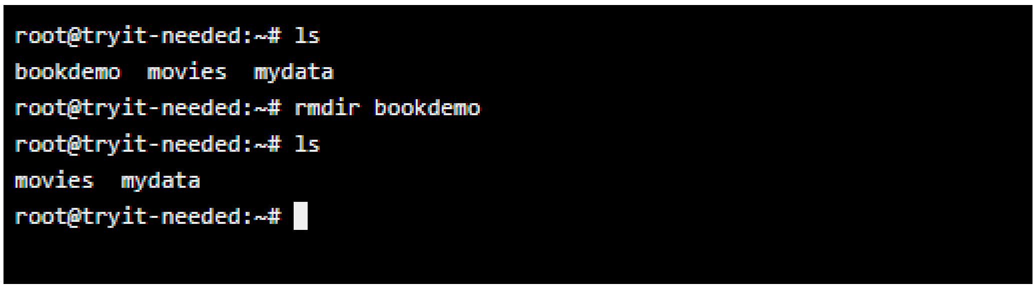
This command is used to remove the directories. Also, command allows to delete the multiple directories using a single command but provided all the directories does not contain any file. If the directory contain the files, it will generate the error message like “Directory is not an empty”.

The parent directory cannot be removed if the sub directories exist. First the sub directories has to be removed and after that parent directory can be removed.

**Syntax :**

```
rmdir [options] [directoryname(s)]
```

**Example :**



```
root@tryit-needed:~# ls
bookdemo  movies  mydata
root@tryit-needed:~# rmdir bookdemo
root@tryit-needed:~# ls
movies  mydata
root@tryit-needed:~#
```

**rmdir –v**: This option displays verbose information for every directory being processed.

## 2.5 USERS AND GROUPS

Linux operating systems have the ability to multitask in a manner similar to other operating systems. However, Linux's major difference from other operating systems is its ability to have multiple users. Linux was designed to allow more than one user to have access to the system at the same time. In order for this multiuser design to work properly, there needs to be a method to protect users from each other. This is where permissions come in to play.

### User

A system user or an account is uniquely identified which is called the UID (unique identification number). There are two types of users – the root or super user and normal users. A root or super user can access all the files, while the normal user has limited access to files. A super user can add, delete and modify a user account.

A user can be added by running the useradd command at the command prompt. After creating the user, set a password using the passwd utility, as follows:

```
[root@localhost~:]# useradd tom
[root@localhost~:]# passwd tom
Changing password for user tom.
New password:
```

```
Retype new password:
Passwd: all authentication token updated successfully
```

**Group**

A mechanism to organize a collection of users refers to a group in Linux. Like the user ID, each group is also associated with a unique ID called the GID (group ID). There are two types of groups – a primary group and a supplementary group. Each user is a member of a primary group and of zero or 'more than zero' supplementary groups.

You can add a new group with default settings by executing the groupadd command as a root user.

```
[root@localhost:~]# groupadd professor
```

If you wish to add a password, then type gpasswd with the group name, as follow:

```
[root@localhost:~]# gpasswd professor
Changing the password for group professor

New Password:
Re-enter new password:
```

## 2.6 LET US SUM UP

**Points to ponder**

- A file system is a logical collection of files on a partition or disk.
- A partition is a container for information and can span an entire hard drive if desired.
- Linux/UNIX treats all the physical devices as files only
- All files and file system objects are ordered within a hierarchical file tree with exactly one root directory '/'.
- A system user or an account is uniquely identified which is called the UID (unique identification number).
- A mechanism to organize a collection of users refers to a group in Linux
- Each group is associated with a unique ID called the GID (group ID)

## 2.7 CHECK YOUR PROGRESS

**State true or false**

1) The ordinary user cannot create a new system user or account.
2) The –b option is used to remove a directory including all its subdirectories.
3) ls –R display all the files in your current directory and its subdirectories including the hidden files.
4) cd command is use to move from the current working directory to other specified directory
5) Physical devices in Linux are not treated as files.
6) cat command is only used for creating a new files.
7) cp command cannot be applied on the group of files
8) cp with option –i refers to the interactive copy.
9) rm command is used for deleting the directories
10) pwd command can only print the actual path.

**Give the name of the command for the following task.**

1) Moving the files from one directory to another
2) Making a clone of the file or folder
3) Viewing the contents of the file
4) Creating the directory with specified permissions
5) Viewing the file permission of a specific file
6) Deletion of a file forcefully
7) Renaming a file
8) Deleting the directory and files contained in that directory
9) Adding the user
10) Set only execute permission for the user, group and others

**Match the following**

| | |
|---|---|
| 1) chmod | a) cut and paste |
| 2) mv | b) permission |
| 3) cp | c) new user addition |
| 4) adduser | d) clone |
| 5) ls | e) file details |

## 2.8 FURTHER READING

1. Here is an article about the various Linux commands: https://www.geeksforgeeks.org/linux-commands/

## 2.9 ASSIGNMENTS

1) Discuss in detail, the working of the chmod command for setting the permission of files and directories.
2) Explain the different use of the ls command in Linux

## 2.10 ACTIVITIES

**Q1. Execute the following commands:**

1) Display the list of files contained in the home directory
2) Copy files from your current directory to the one step previous directory
3) Remove all the files and folders of your current directory
4) Set all the access rights to your any one of the directory
5) Display all file having an extension ".txt"

## 2.11 Case Studies

1) Prepare a detail chart of various basic Linux commands as a quick reference manual.

# Unit 3: General Purpose Utilities Commands

3

## Unit Structure

3.1. Learning Objectives

3.2. Introduction

3.3. General Purpose Utilities Commands

3.4. Let Us Sum Up

3.5. Check Your Progress

3.6. Further Reading

3.7. Assignments

3.8. Activities

3.9. Case Studies

## 3.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- use the general purpose commands
- record their terminal session
- use the calculator utility in Linux

## 3.2 INTRODUCTION

This unit discusses the general purpose utilities available in UNIX based Linux operating system. The commands like clear command for clearing the console terminal screen, script and scriptreplay command for recording the commands and terminal activities and also history command for storing previous command used are discussed in the following subsection.

## 3.3 GENERAL PURPOSE UTILITIES COMMANDS

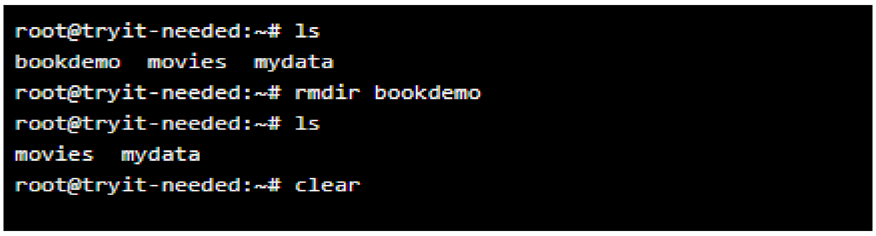
### 1. clear COMMAND

This is a standard UNIX computer operating system command used to clear the visible area of the console terminal screen. This command first looks for a terminal type in the environment and after that, it figures out the terminfo database for how to clear the screen. And this command will ignore any command-line parameters that may be present. Also, the clear command doesn't take any argument and it is almost similar to cls command on a number of other operating systems.

**Syntax:**

```
clear
```

**Example:**



```
root@tryit-needed:~# ls
bookdemo  movies  mydata
root@tryit-needed:~# rmdir bookdemo
root@tryit-needed:~# ls
movies  mydata
root@tryit-needed:~# clear
```

```
root@tryit-needed:~#
```

As we write the clear command, the terminal will clear the contents.

## 2. Echo COMMAND

Echo is one of the most commonly and widely used built-in command for Linux bash and C shells, that typically used in scripting language and batch files to display a line of text/string on standard output or a file.

**Syntax**:

echo [OPTION]... [STRING]...

**Example**:

echo "Welcome to BabaSaheb Ambedkar Open University"

**Output**:

BabaSaheb Ambedkar Open University

**Options**

**\b :** it removes all the spaces in between the text

**Example**:

```
echo –e "Wel \bcome \bnewbies"
Welcomenewbies
```

**\n :** this option creates new line from where it is used

```
echo –e "Welcome \nnewbies"
Welcome
newbies
```

**\t :** this option is used to create horizontal tab spaces

```
echo –e "Welcome \tnewbies"
Welcome      newbies
```

**\v :** this option is used to create vertical tab spaces.

```
echo –e “Welcome \vnewbies”
Welcome
newbies
```

**echo * :** this command will print all files/folders, similar to ls command.

**bc - Unix calculator**

bc is a command which provides facilities like calculator. It stands for bench calculator. It can be used as an interactive mathematical shell or can be treated like a programming language.

Syntax:

```
bc [-c] [-l] [file]
```

options

| | |
|---|---|
| -c | stands for compile only |
| -l | it defines the math functions |
| file | file containing the bc commands |

**Available operators:**

| | |
|---|---|
| + | Addition |
| - | subtraction |
| * | Multiplication |
| / | Division |
| ^ | power (raise to) |
| % | remainder |
| ++ | prefix |
| -- | Postfix |

==, <=, >=, <> etc., assignment operators

Example

```
[user@localhost:~]# bc
```

**output**

```
Ibase=10

Obase=2

5

101
```

In above example ibase is input base, where 10 means input values is in decimal and obase indicates the output base is in binary. So if input value is 5 then its output is binary=101

```
13 + 6

19

6/4

1

Scale= 2

7/2

3.5
```

The addition, subtraction, multiplication can be carried out using the bc command.

By default the scale value is 0.

## 3. script-MAKE TYPESCRIPT OF TERMINAL SESSION

Script command is used makes a typescript of everything printed on your terminal.

Script tag is useful as hard copy which work as interactive session. It save all data in file if filename is passed as second argument. It is useful for users who need a hardcopy record of an interactive session as proof of work done, as the typescript file can be printed out later with lpr. If the file argument is given, script saves all dialogue in file. If no file name is given, the typescript is saved in a file named typescript.

**Syntax**:

script [-a] [-c command] [-e] [-f] [-q] [-t[=file]] [-V] [-h] [file]

**Options**:

| | |
|---|---|
| -a | It appends the output content to the existing file or typescript and retains the prior contents |
| -q | It stands for quiet. It does not display any output |
| -f | It stands for flush. It flushes the output after each write. This option is useful for telecooperation in which one user does "mkfifo temp; script -f temp", and another can supervise real-time what is being done using "cat temp". |
| -c | It executes the command rather than an interactive shell. |
| -t | It outputs the timing data to standard error or to a specified file. This data comprises of two fields which are separated by a space. The first indicates how much time elapsed since the previous output while second indicates the number of characters output at that time,For replaying the typescript with realistic output delay and typing. |

The script ends when the forked shell exits (a control-D to exit the Bourne shell (sh), and exit, logout or control-d (if ignore eof is not set) for the C-shell, csh).

Certain interactive commands, such as vi, create garbage in the typescript file. script works best with commands that do not manipulate the screen; the results are meant to emulate a hardcopy terminal.

**Example:**

script log.txt

It starts logging all the results of the command in the file named log.txt. For logging, this command opens a subshell and records all the outputs. The scripts can be ended by pressing control key+ D or when user types exit at the terminal.

The output will be written to the file log.txt and the last line of the file will log the date and time the command was executed.

script –c “ps ax” proc.txt

This command with **a** and**x** options forces to list all the process currently on the system which user have access to see. Script command here logs the output of ps command in the file proc.txt, and the last line of the file will log the date and time the command was executed.

```
script –c “echo hello” welcome.txt
```

Logs the output of the echo command that echoes the text "hello,". Since the entire command must appear in quotes. The output will be written to the file hello.txt, and the last line of the file will log the date and time the command was executed.

**Recording several terminal sessions**

We can record several terminal session. When you finish a record, just begin another new session record. It can be helpful if you want to record several configurations that you are doing to show it to your team or students for example. You just need to name each recording file.

```
# script configuration1_record
  ………………………………
          Configuration step
  ………………………………
# exit
```

Once you finished with first configuration, begin to record the next configuration. And so on for the other.

```
# script configuration2_record
  ………………………………
          Configuration step
  ………………………………
# exit
```

**Replay a Linux terminal session**

Up to now we have seen the content of the recorded file with commands to display a text file content. The script command also gives the possibility to see the recorded session as a *video.* It means that you will review exactly what you have done step by step at the moment you were entering the commands as if you were looking a video. So you will playback/replay the recorded terminal session.

To do it, you have to use --timing option of script command when you will start the record.

```
# script --timing=file_rec_time shell_record1

Script started, file is shell_record1
```

See that the file into which to record is shell_record1. When the record is finished, exit normally

```
# exit

exit

Script done, file is shell_record1
```

Let's see check the content of file_rec_time.

```
# cat file_time

0.607440 49

0.130061 1

126.131648 1

0.126914 1

0.133997 1
```

```
0.126936 1

0.114201 1

0.382766 1

0.321079 1

0.112105 2

0.363375 152
```

The --timing option outputs timing data to the file indicated. This data contains two fields, separated by a space which indicates how much time elapsed since the previous output how many characters were output this time. This information can be used to replay typescripts with realistic typing and output delays.

Now to replay the terminal session, we use scriptreplay command instead of script command with the same syntax when recording the session. Look below

```
# scriptreplay --timing= file_rec_time shell_record1
```

You will that the recorded session with be played as if you were looking a video which was recording all that you were doing. You can just insert the timing file without indicating all the --timing= file_rec_time. Look below

```
# scriptreplay file_rec_time shell_record1
```

So you understand that the first parameter is the timing file and the second is the recorded file.

**History Command**:

This command is used by the user to store the previous command used. It differs from the script command, by storing only the command used while script command stores all the activities of the terminal session recorded.

Syntax:

```
history [options]
```

Few options are mentioned below:

| | |
|---|---|
| -c | for clearing the history list |
| -w | write the current history to the history file and append them |
| -a | append the history lines from this session to the history file |

```
$history
1       ls -l
2       who
3       cal 2019
4       clear
5       history
6       clear
```

In output, the numbers preceded before each command refers to the event number which may differ in representation system to system.

```
$ history 4
1       ls -l
2       who
3       cal 2019
4       clear
```

It displays the limited number of commands that executed previously as follows:

```
$ history –d 1
2        who
3        cal 2019
4        clear
```

History can also be removed using history -d event_number

```
$ history –c
```

This will remove the whole history.

## 3.4 LET US SUM UP

**Points to ponder**

- script and scriptreplay commands in Linux helps you to record commands and their output printed on your terminal during a given session.
- history command frequently used in our daily routine jobs to check history of command or to get info about command executed by user.
- bc is a powerful calculator like utility.

## 3.5 CHECK YOUR PROGRESS

**Give the name of the command for the following task.**

1) Viewing the previously executed commands
2) Displaying a line on the terminal
3) Deleting the entire history
4) Performing the addition of two numbers

**Match the following**

| | |
|---|---|
| 1) history | a) recording the terminal activities |
| 2) echo | b) displaying output on standard terminal |
| 3) clear | c) viewing previous commands executed |
| 4) script | d) calculator |
| 5) bc | e) clear the terminal |

## 3.6 FURTHER READING

1) https://www.geeksforgeeks.org/history-command-in-linux-with-examples/
2) https://www.tecmint.com/record-and-replay-linux-terminal-session-commands- using-script/

## 3.7 ASSIGNMENTS

1) Discuss in detail, the working of the script command for logging the activities.
2) Explain the purpose of echo command in Linux.

## 3.8 ACTIVITIES

**Q1. Execute the following commands:**

1) Display the history of previously executed commands
2) Evaluate any mathematical expression using bc utility
3) Delete few entries from history file

Q2. Prepare a log file which maintains all the terminal activities of current session and view the recorded activities.

## 3.9 CASE STUDIES

1) Record a video for installing or configuring any utility in Linux system using scriptreplay command.

# Unit 4: General Purpose Utilities On File

4

## Unit Structure

4.1. Learning Objectives

4.2. Introduction

4.3. General purpose utilities On File

4.4. Let us sum up

4.5. Check your Progress

4.6. Further Reading

4.7. Assignments

4.8. Activities

4.9. Case studies

## 4.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Change the file permissions on files and directories
- Search the files and displaying the contents of the files
- Update a file’s date and time

## 4.2 INTRODUCTION

This unit discusses the general purpose utilities available for files in UNIX based Linux operating system. The commands discussed in the following subsection belongs to the changing the file permissions, searching and displaying the contents of the file(s).

## 4.3 GENERAL PURPOSE UTILITIES ON FILE

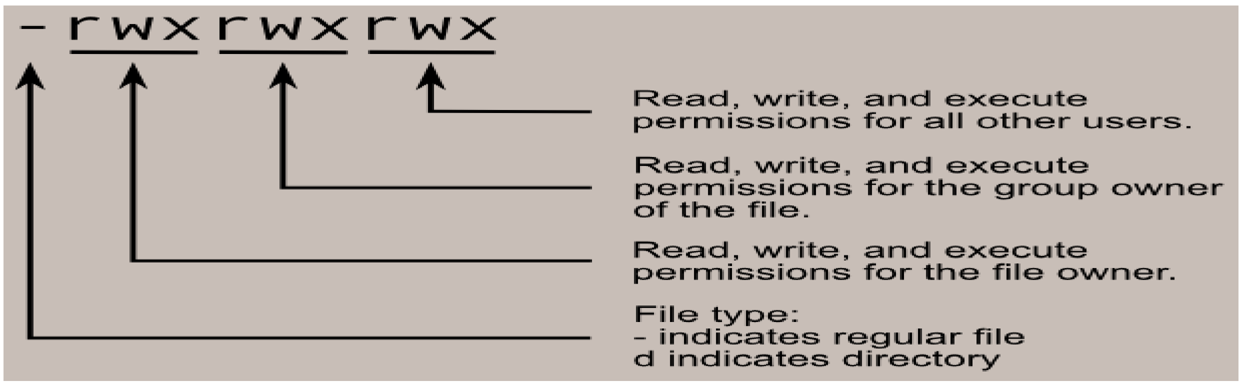
**1. chmod**

chmod changes the permissions of each given file according to mode, where mode describes the permissions to modify. Mode can be specified with octal numbers or with letters.

**Syntax:**

```
chmod options permissions file name
```

File Permission on the Linux refers to the access rights for the owner, the member of the group and everybody else using the file. Each file and directory is assigned the access rights. This can be a read, write or an execute permission on the file.


- rwx rwx rwx
Read, write, and execute permissions for all other users.
Read, write, and execute permissions for the group owner of the file.
Read, write, and execute permissions for the file owner.
File type:
- indicates regular file
d indicates directory

We can see the permission for the files and directories using ls –l command i.e., long listing which is already covered in previous unit. The commands related to the file and directory permissions are discussed here.

chmod: change mode (for files)

The chmod command is used to change the file or a directory permissions. You can specify the desire permission setting for file or files you wish to modify. There are two ways to set the same out of which one is using octal notations.

Here's how it works:

| | |
|---|---|
| Read permission (r) | =4 |
| Write permission (w) | =2 |
| Execute permission(x) | =1 |

If user wish to set all the access rights on a particular file for user only then it will be be the addition of (4+2+1) i.e., rwx . This is a convenient way to represent each of the three sets of permission (owner, group, and other) as a single digit.

In this example permission for group and everybody else will be 0

**Example**

```
chmod 700 temp.txt

$ls –l temp.txt

-rwx------ 10 user1 group1 2019 Apr 12 05:12 temp.txt

chmod 740 temp.txt

$ls –l temp.txt

-rwxr----- 10 user1 group1 2019 Apr 12 06:14 temp.txt

chmod 771 temp.txt

$ls –l temp.txt
```

```
-rwxrwx--x 10 user1 group1 2019 Apr 12 06:14 temp.txt
```

chmod: change mode (for directories)

The chmod command can also be used to control the access permissions for directories. Again, we can use the octal notation to set permissions, but the meaning of the r, w, and x attributes is different:

- r - Allows the contents of the directory to be listed if the x attribute is also set.
- w - Allows files within the directory to be created, deleted, or renamed if the x attribute is also set.
- x - Allows a directory to be entered (i.e. cd dir).

**Here are some useful settings for directories:**

| Value | Meaning |
|---|---|
| 777 | (rwxrwxrwx) No restrictions on permissions. Anybody may list files, create new files in the directory and delete files in the directory. Generally not a good setting. |
| 755 | (rwxr-xr-x) The directory owner has full access. All others may list the directory, but cannot create files nor delete them. This setting is common for directories that you wish to share with other users. |
| 700 | (rwx------) The directory owner has full access. Nobody else has any rights. This setting is useful for directories that only the owner may use and must be kept private from others. |

**2. chown**

It stands for change ownership. As in Linux all files are associated with a group and an owner, the chown command helps to change the user and group ownership of any given file and directory in a Linux file system.

**Syntax:**

```
chown [options] USER [:GROUP] FILE(s)
```

Where user is refers to the user ID (UID) of the new owner, group refers to the new group (GID) and FILE(s) is the name of one or more files, directories or links.

**Changing the owner of the file**

-new owner's name and the target file is to be passed as an argument in the chown command

```
chown user filename
```

**Example:**

```
$ chown linuser f1.txt
```

The above command will change the owner of the file f1.txt to a new owner named linuser.

If we want to change the ownership of more than one file or directory, the file or directory names are separated by the space.

**Example**

```
$ chown linuser f1.txt dir1
```

The above command will change the ownership of the file f1.txt and directory d1 by a new owner linuser.

Instead of user name, we can use the UID i.e., User ID

**Example**

```
$ chown 1100 f1.txt dir1
```

We can also change the owner and group of file by the chown command followed by new owner and group separated by a colon (:) with no space in between and the target file.

```
chown USER:GROUP filename
```

**Example:**

```
$ chown linuser:user f1.txt
```

The above command will change the ownership of the file f1.txt by a new owner linuser and group user.

If you omit the group name after the colon (:) the group of the file is changed to the specified user's login group.

```
$ chown linuser: f1.txt
```

To recursively operate on all files and directories under the input directory –r option is used.

```
$ chown –R USER:GROUP DIRECTORY
```

**Example**

```
$chown pri-data: /home/linuser
```

The above command will change the ownership of all the files and directories under /home/linuser directory to a new owner and group named pri-data.

**3. chgrp:** changes groupownership of a file or files.

This functionality also can be performed by the chown command as mentioned above. The owner is set by the chown command and the group by the chgrp command in linux.

**Syntax :**

```
chgrp [option] group file
chgrp [option] –reference=RFILE
```

Before using this command, you should know the users existing on your system, otherwise it will fire an error of an invalid user

To view the list of the users, you may use (cat/etc/group). As mentioned earlier, we can also add a particular user by the following command and then use the chgrp command accordingly.

```
sudo addgroup username_to_be_added
```

1) changing the group name of a file

Before that let us see, the user and group details.

```
$ ls –l

-rw-rw-r--1 user user 0 apr 11 14:22 f1.txt
```

Now apply the chgrp command,

```
$chgrp linuser f1.txt
```

To verify the same, we can apply the ls –l command to view the changes,

```
$ ls –l

-rw-rw-r--1 user linuser 0 apr 11 14:27 f1.txt
```

Here, we can see that group name is changed to linuser.

2) Changing the group name of the folder (directory)

```
$ ls –l

drw-rw-r--1 user user 4096 apr 11 15:01 dir1
```

Now apply the chgrp command,

```
$chgrp linuser dir1

$ ls –l

drw-rw-r--1 user linuser 4096 apr 11 15:04 dir1
```

Here, we can see that group name for directory dir1 is changed to linuser.

Note: -R option can be used for the recursive changes in the folder.

## 4. Find

Thiscommand is used to find files and directories. It permits searching by file, folder, name, and date of creation, date of modification, permissions and owner.The general syntax is given as follows:

```
find [path from where searching starts] [expression] [-options] [what to find]
```

1) Searching a specific file

```
$find ./temp –name log.txt
```

This will search the file log.txt in temp directory. The –name command is case sensitive.

```
$find ./temp –iname log.txt
```

This will search the file log.txt in temp directory but –iname command ignores the case of your query.

```
$find / -iname log.txt
```

Here / modifies indicates that command will search all directories starting from the root directory.

2) Searching a file with pattern

```
$ find ./tmp –name *.txt
```

This will display all the files having .txt as an extension at the end.

3) Searching for empty files and directories

```
$ find ./tmp –empty
```

This command find all empty folders and files in the entered directory or sub-directories.

4) Search for file with entered permissions

```
$ find ./tmp –perm 721
```

This command find all the files and folders in the temp directory having the permission 721.

5) Finding specific type of results.

```
$find ./tmp –type f –iname log.txt
```

This command will search for regular files by supplying –type f modifier. Similarly, we can use d for directory, l for symbolic links, c for character devices and b for block devices.

## 5. Head

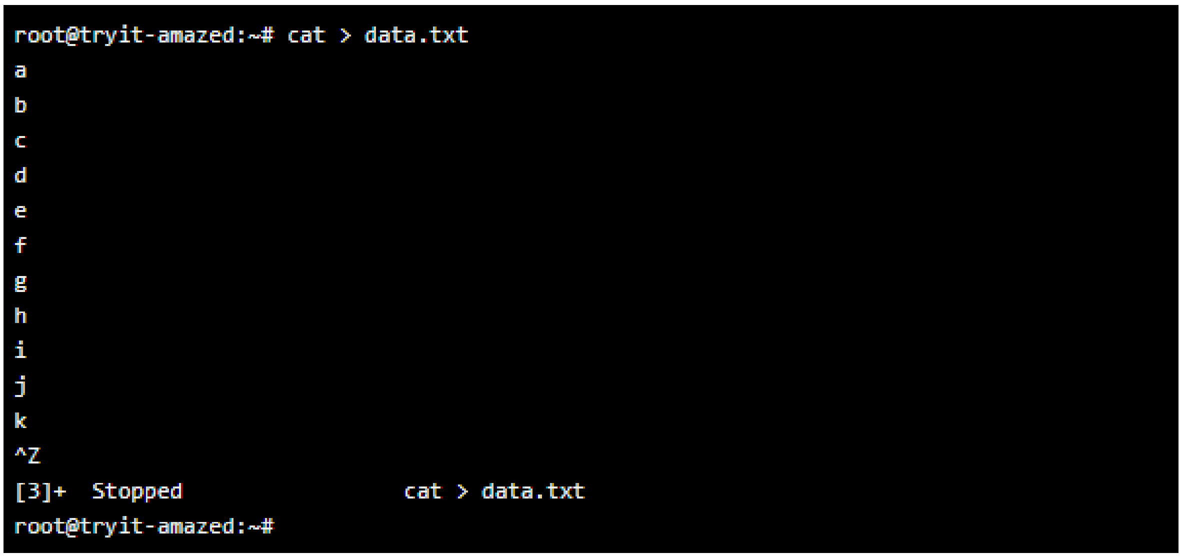
Head command is use to display the top N specified content of the file. It displays the first 10 lines by default.

**Syntax:**

    Head [option] [file]

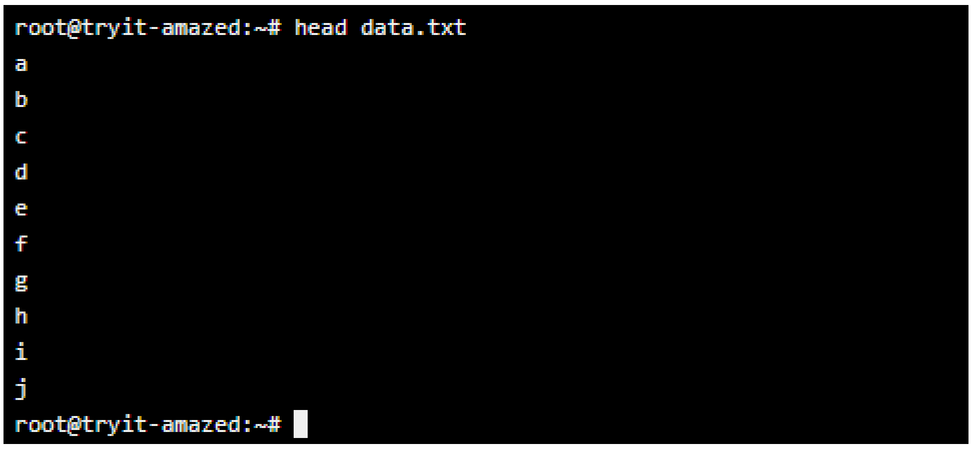
**Example:**

To use head command first create a file as shown below:



```
root@tryit-amazed:~# cat > data.txt
a
b
c
d
e
f
g
h
i
j
k
^Z
[3]+  Stopped                 cat > data.txt
root@tryit-amazed:~#
```

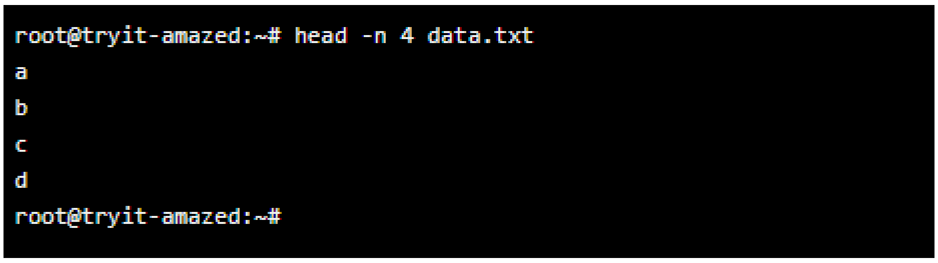
Now apply head command and it will display the starting 10 lines of filename "data.txt".



```
root@tryit-amazed:~# head data.txt
a
b
c
d
e
f
g
h
i
j
root@tryit-amazed:~#
```

**Options:**

-n num Prints the first 'num' lines instead of first 10 lines. Num is mandatory to be specified in command otherwise it will show an error.



```
root@tryit-amazed:~# head -n 4 data.txt
a
b
c
d
root@tryit-amazed:~#
```

-c num Prints the first 'num' bytes from the specified file.

**Use of head command with pipeline (|)**

If we want to print most recently used file then we can use ls command along with head command by means of pipeline.

```
$ls –t
```

It displays the recently modified files or directories.

```
$ ls –t | head –n 2
```

It will display the two most recently modified files.

```
$ ls –t | head –n 2 | sort
```

It will display the two most recently used files or directories in the alphabetical order.

There are number of other filters or commands along which we use head command. Mainly, it can be used for viewing huge log files in UNIX.

## 6. Tail

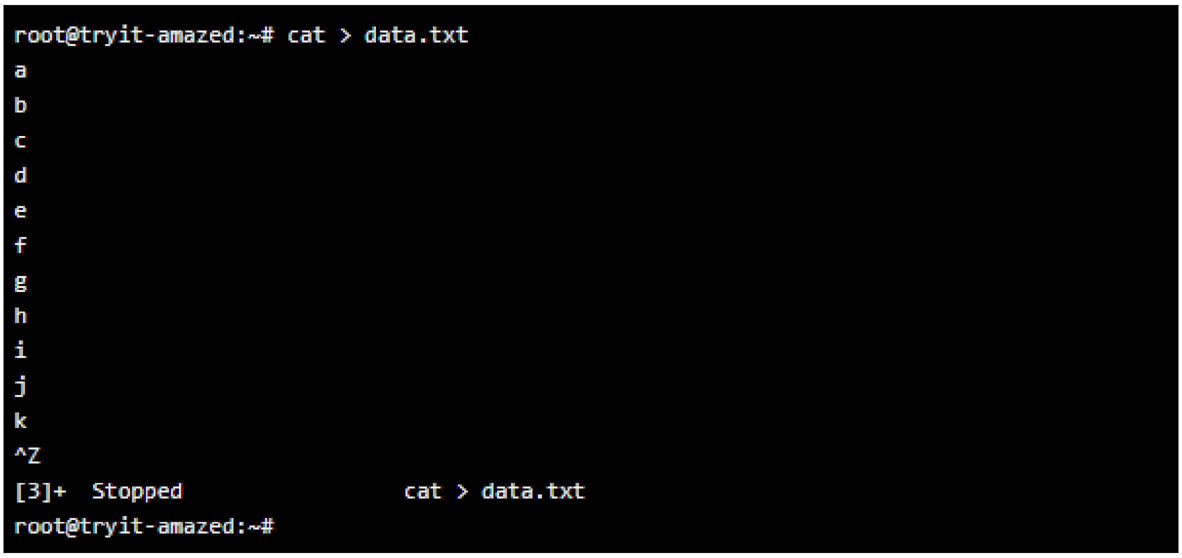
Tail command is use to display the last N specified contents of the file. It displays the last 0 lines by default.

**Syntax :**

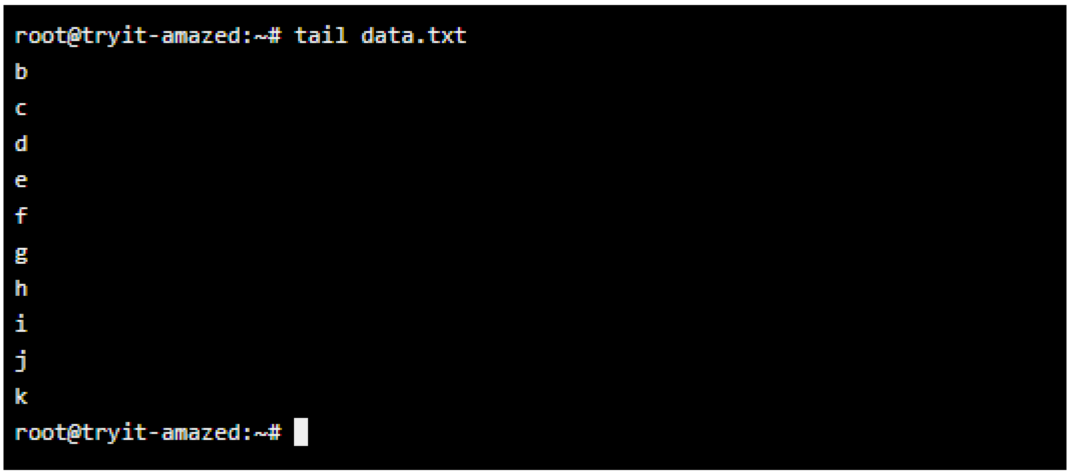
```
tail [option] [file]
```

**Example :**

To use tail command first create a file as shown below:



```
root@tryit-amazed:~# cat > data.txt
a
b
c
d
e
f
g
h
i
j
k
^Z
[3]+  Stopped                 cat > data.txt
root@tryit-amazed:~#
```



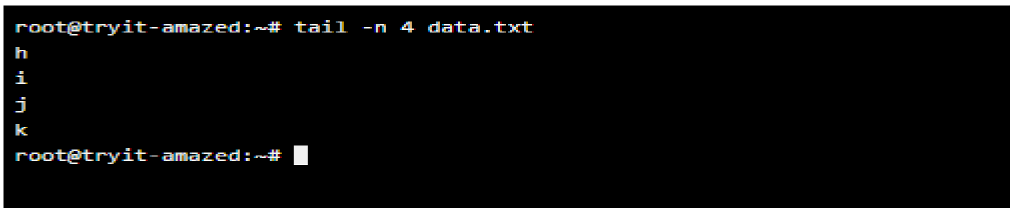
```
root@tryit-amazed:~# tail data.txt
b
c
d
e
f
g
h
i
j
k
root@tryit-amazed:~#
```

**Options:**

-n num Prints the first 'num' lines instead of first 10 lines. Num is mandatory to be specified in command otherwise it will show an error.

Now apply tail command and it will display the last 10 lines of filename "data.txt".

-c num Prints the last 'num' bytes from the specified file.



```
root@tryit-amazed:~# tail -n 4 data.txt
h
i
j
k
root@tryit-amazed:~#
```

**Use of tail command with pipeline (|)**

This command also can be piped with many other commands of the unix.

**Example**:

$ tail -n 6 states.txt

Sikkim

Tamil Nadu

Uttar Pradesh

West Bengal

$ tail -n 7 state.txt | sort -r

West Bengal

Uttar Pradesh

Tamil Nadu

Sikkim

The output of tail command is piped to sort command. The last 4 lines of the states.txt will be sorted in reverse order. Also this can be piped with one or more filters for the additional processing.

**Example:**

$ cat states.txt | head -n 10 | tail -n 5 >out.txt

Let us view the contents of out.txt file:

**$ catout.txt**

Manipur

Meghalaya

Mizoram

Nagaland

Odisha

**Let us understand the working of the command:**

- First of all, **cat** command fetch the contents present in the file named states.txt
- After that output of cat command is piped to the **head** command.
- Head command fetch all the data from start (line number 1) to the line number 10 and the output of **head** command is piped to **tail** command.
- Now, tail command displays the last 5 lines of the data and the output is redirected to a file name **out.txt** via directive operator.

## 7. wc (word count)

As it name signifies, it is mainly used for counting purpose. It is used for counting number of lines, byte, character and word count for the specified file passed as an argument.

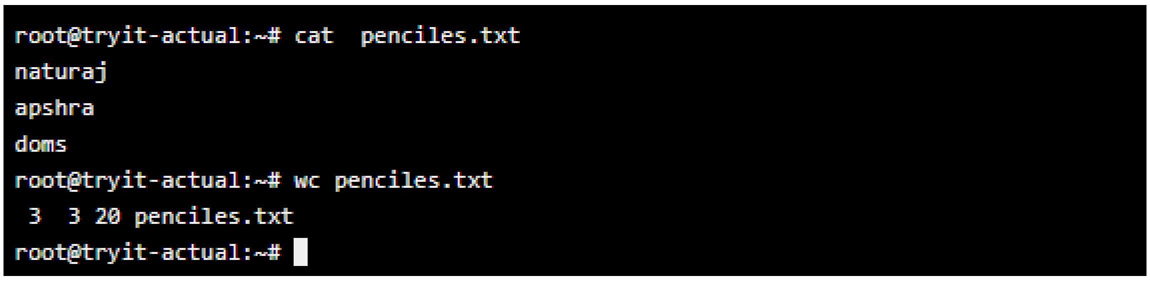
wc by default it display four-column output:

1. First column shows number of lines in a file.
2. Second column shows number of words in file.
3. Third column shows number of characters.
4. Fourth column display file name.

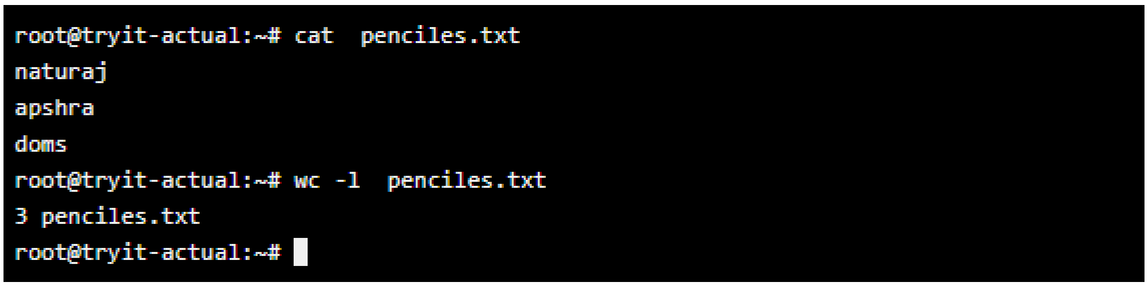
**Syntax:**

```
wc [options][file]
```

**Example:**

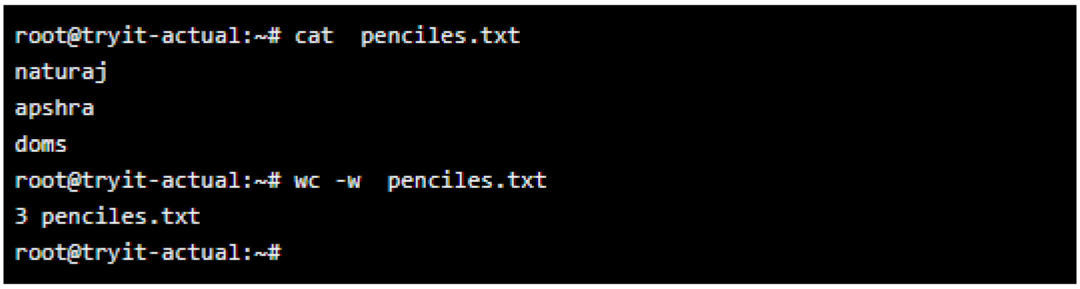


```
root@tryit-actual:~# cat  penciles.txt
naturaj
apshra
doms
root@tryit-actual:~# wc penciles.txt
 3  3 20 penciles.txt
root@tryit-actual:~#
```



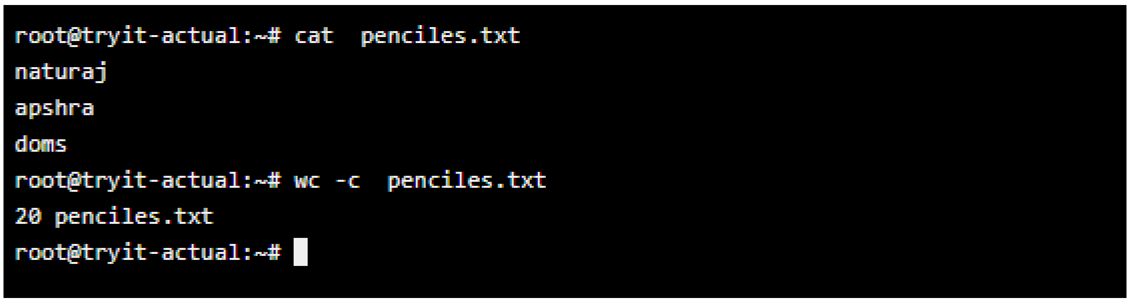
```
root@tryit-actual:~# cat  penciles.txt
naturaj
apshra
doms
root@tryit-actual:~# wc -l  penciles.txt
3 penciles.txt
root@tryit-actual:~#
```

-l option prints the number of lines present in a file.



```
root@tryit-actual:~# cat  penciles.txt
naturaj
apshra
doms
root@tryit-actual:~# wc -w  penciles.txt
3 penciles.txt
root@tryit-actual:~#
```

-w option prints the **number of words** present in a file.



```
root@tryit-actual:~# cat  penciles.txt
naturaj
apshra
doms
root@tryit-actual:~# wc -c  penciles.txt
20 penciles.txt
root@tryit-actual:~#
```

-c option prints the **count of bytes** present in a file.

This command can be used to count the files and folders present in current directory when it is piped with ls command

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

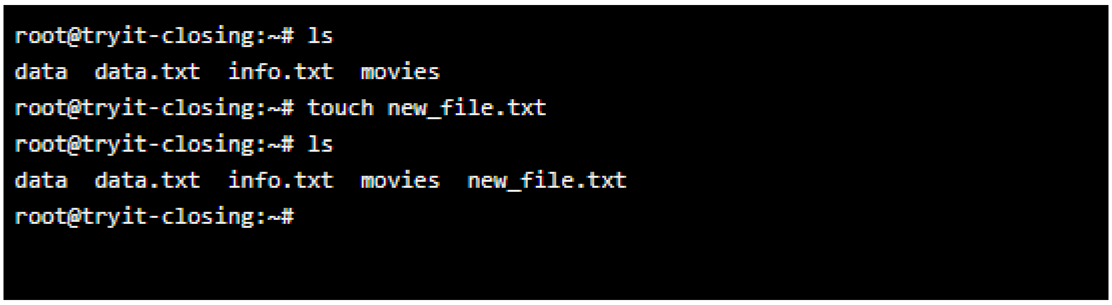
8. **Touch**

Touch command is use to create an empty file. It differs from the cat command in which file is created with contents. It is also used to change and modify the timestamps of a file.

**Syntax :**

```
touch<filename>
```

**Example :**



```
root@tryit-closing:~# ls
data  data.txt  info.txt  movies
root@tryit-closing:~# touch new_file.txt
root@tryit-closing:~# ls
data  data.txt  info.txt  movies  new_file.txt
root@tryit-closing:~#
```

We can also create multiple files using touch command.

touch file1, file2, file3

**Options:**

| | |
|---|---|
| -a | it is used to change access time only |
| -c | it is used to check whether file is created or not. It avoids creating file if not created. |
| -c-d | it is used to modification and access time |
| -m | it is used to change the modification time only |
| -t | it is used to create a file using a specified time |

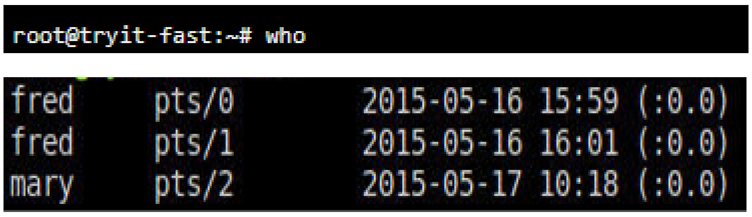
**9. Who**

The who command prints information about all users who are currently logged in. It is also used to find out when the system booted last and also the current run level of the system.

**Syntax :**

**who [options][filename]**

**Example :**



```
root@tryit-fast:~# who
```

```
fred     pts/0        2015-05-16 15:59 (:0.0)
fred     pts/1        2015-05-16 16:01 (:0.0)
mary     pts/2        2015-05-17 10:18 (:0.0)
```

i.e., if no options are specified than it will display the login name of the user, terminal line numbers, and login time of the users as above.

**Options**

| | |
|---|---|
| -a | displays all the details of current logged in user |
| -r | display the current run level of the system |
| -l –H | display the system login process details |
| -q –H | count the number of users logged on |
| -u | displays the list of users currently logged on |

## 4.4 LET US SUM UP

**Points to ponder**

- File Permission on the Linux refers to the access rights for the owner, the member of the group and everybody else using the file
- The mode in chmod command can be specified with octal numbers or with letters
- In Linux all files are associated with a group and an owner

## 4.5 CHECK YOUR PROGRESS

**Give the name of the command for the following task.**

1) Counting the number of words in a given specified file.
2) Creating an empty file.
3) Changing the group of a file
4) Creating the directory with specified permissions
5) Change the owner of a file
6) Displaying the user's information who are logged on to the system.
7) Display the last 20 lines of a file in a sorted order
8) Redirecting the top 20 lines of a given file in a new file
9) Change the timestamp of  given file
10) Search a specific file

**Match the following**

|  |  |
|---|---|
| 1) touch | a) changing the owner |
| 2) chown | b) creating empty file |
| 3) > | c) user login information |
| 4) who | d) pipe |
| 5) \| | e) redirection |

## 4.6 FURTHER READING

1) Refer the following for chmod command:
   https://www.computerhope.com/unix/uchmod.htm

## 4.7 ASSIGNMENTS

1) Discuss in detail, the working of the chmod command for setting the permission of files and directories.
2) Explain the different use of the ls command in Linux
3) Discuss any application where head and tail command can be used.
4) Explain pipeline and redirection operator.

## 4.8 ACTIVITIES

**Q1. Execute the following commands:**

```
who –l -H
who –u
ls –l | wc –l
ls | wc –l
touch –a temp
```

Q2.All the student login in to the Linux system by their respective user id and password. You are supposed to display the information of all active users and count the number of users and check their activities.

Also try all the different options available with who command and record its output

## 4.9 CASE STUDIES

1) **Study the following commands for searching:**

```
grep
locate
```

Also, compare and contrast it with find command.

# Block-4

# Process, File Management & Shell Programming in Linux/Unix

# Unit 1: Process Management and Related Commands

1

## Unit Structure

1.1. Learning Objectives

1.2. Introduction

1.3. Process Status and Background and Foreground Processes

1.4. Jobs Execution With Priorities (nice Command)

1.5. Termination Of Job (kill Command)

1.6. Scheduling Jobs and Running Jobs in Periodically

1.7. Wait And Sleep Commands

1.8. Let Us Sum Up

1.9. Further Reading

1.10. Assignments

1.11. Activities

1.12. Case Studies

## 1.3 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Run any process at foreground or at background
- Schedule process to run at particular time
- Terminate process
- Set priority to processes
- Periodically run processes

## 1.4 INTRODUCTION

Process management in Linux is tricky Business. Linux creates a process whenever a program is launched, either by you or by Linux. This process is a container of information about how that program is running and what's happening.

If the process runs and terminates correctly, then everything is fine; however, if it hogs the CPU, or refuses to go when its time is up, then the Linux commands described below may help you to restore control over them.

Let's start with a list of things you may want to do when managing Linux processes:

- See which processes are running
- See how much of your Linux system the processes are using (especially any greedy ones)
- Locate a particular process to see what it's doing or to take action on it
- Define or change the level of priority associated with that process
- Terminate the process if it has outlived its usefulness or if it's misbehaving

The commands described below should be entered via the command line interface. Simply open a terminal (all-text) window to access this interface. It may look basic, but it's actually very powerful and flexible – just the thing for keeping all those processes in line.

## 1.3 PROCESS STATUS AND BACKGROUND AND FOREGROUND PROCESSES

### ps :

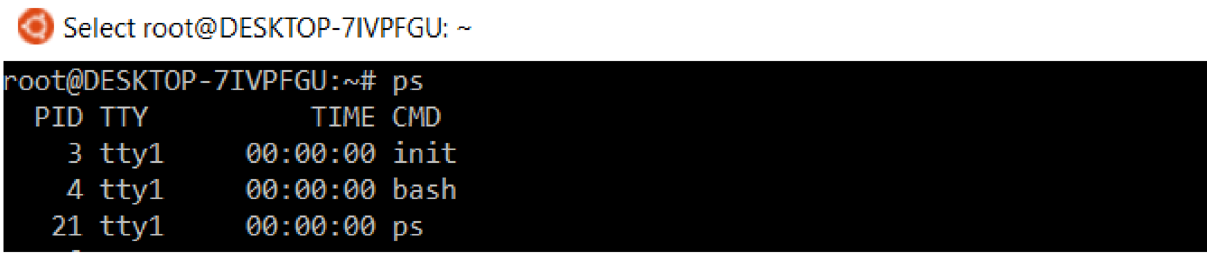
The ps command is used for viewing currently running processes on the system.It helps us to determine which process is occupying how much CPU space , doing what , how much memory it is using , process ID, command name, etc .

Though different systems may running different processes at any time, it may display different output in different systems.

**Syntax :**

```
ps
```

**Example :**



Select root@DESKTOP-7IVPFGU: ~

```
root@DESKTOP-7IVPFGU:~# ps
  PID TTY          TIME CMD
    3 tty1     00:00:00 init
    4 tty1     00:00:00 bash
   21 tty1     00:00:00 ps
```

Where

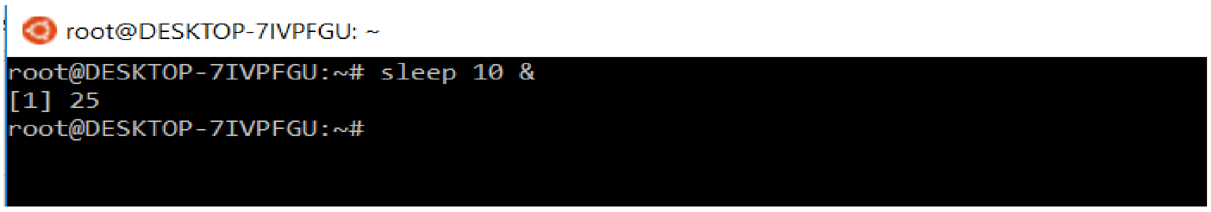
PID is process id

CMD is command or process name which is running

**Run any process in background**

To run any process or command in background we just need to write & at the end of the command or process.

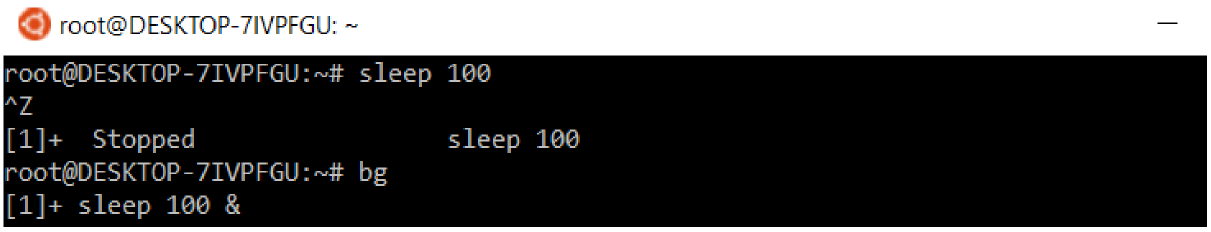
The & will run process in background and give us prompt straight away , allowing us to run another program while waiting for a task to finish.



root@DESKTOP-7IVPFGU: ~

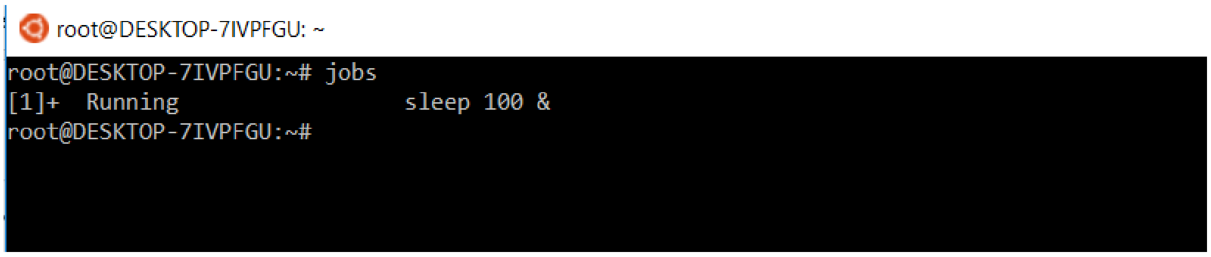
```
root@DESKTOP-7IVPFGU:~# sleep 10 &
[1] 25
root@DESKTOP-7IVPFGU:~#
```

For Backgrounding current running process press **^z (control z)** and thentype **bg.**


```
root@DESKTOP-7IVPFGU: ~
root@DESKTOP-7IVPFGU:~# sleep 100
^Z
[1]+  Stopped                 sleep 100
root@DESKTOP-7IVPFGU:~# bg
[1]+ sleep 100 &
```

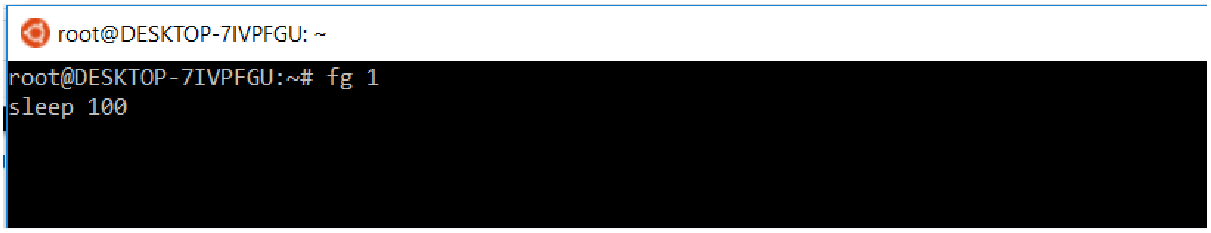
**NOTE :** Do not background interactive process.

To see which processes are running in background type **jobs**


```
root@DESKTOP-7IVPFGU: ~
root@DESKTOP-7IVPFGU:~# jobs
[1]+  Running                 sleep 100 &
root@DESKTOP-7IVPFGU:~#
```

Foregrounding background process

```
fg job_number
```


```
root@DESKTOP-7IVPFGU: ~
root@DESKTOP-7IVPFGU:~# fg 1
sleep 100
```

If we type only fg then it would foreground last running process.

**For Mind-mapping**

| | |
|---|---|
| **command &** | run command in background |
| **^C** | kill the job running in the foreground |
| **^Z** | suspend the job running in the foreground |
| **Bg** | background the suspended job |

|  |  |
|---|---|
| **Jobs** | list current jobs |
| **fg %1** | foreground job number 1 |
| **kill %1** | kill job number 1 |
| **Ps** | list current processes |
| **kill 26152** | kill process number 26152 |

## 1.4 JOBS EXECUTION WITH PRIORITIES (nice COMMAND)

By default, all the processes are considered equally urgent and are allotted the same amount of CPU time. To enable the user to change the relative importance of processes, Linux associates a priority parameter with each job that can be set or changed by the user. The Linux kernel then reserves CPU time for each process based on its relative priority value.

The nice parameter is used for this purpose. It ranges from minus 20 to plus 19 and can take only integer values. A value of minus 20 represents the highest priority level, whereas 19 represents the lowest. The fact that the highest priority level is indicated by the most negative number is somewhat counterintuitive. However, running at a lower priority is considered "nicer," because it allows other processes to use a bigger share of CPU time.

### nice :

It starts new process (jobs) and assign priority (nice) value at the same time. It puts particular process at higher or lower priority than others.

Process priority values range from **-20 to 19**.

A process with the nice value of -20 is considered to be on top of the priority. And a process with nice value of 19 is considered to be low on the priority list.

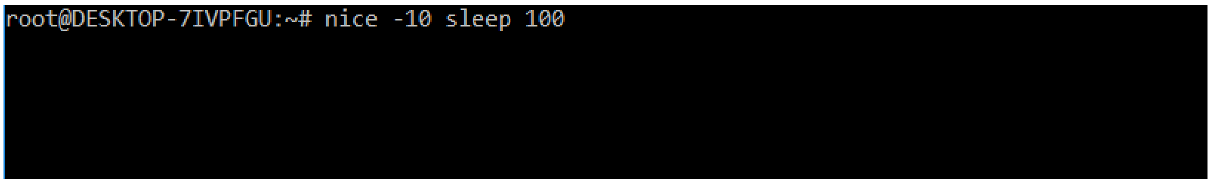
**Syntax :**

```
nice [-n] process_name
```

**Example :**

For example, the following command starts the process and setting the nice value to 10:

**nice -10 process_name**

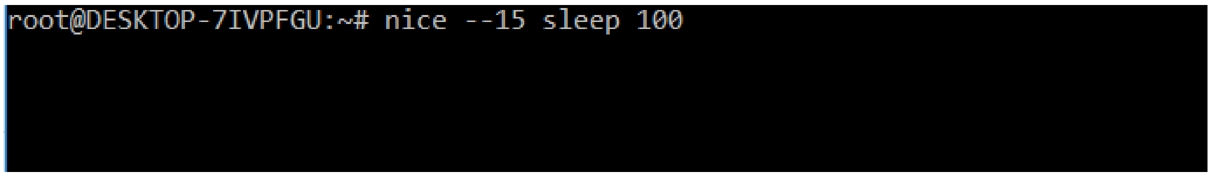


```
root@DESKTOP-7IVPFGU:~# nice -10 sleep 100
```

Note that the dash in front of the 10 does not represent a minus sign. It has the usual function of marking a flag passed as an argument to the nice command.

To set the nice value to minus 10, add another dash:

**nice --15 process_name**



```
root@DESKTOP-7IVPFGU:~# nice --15 sleep 100
```

Remember that lower nice values correspond to a higher priority. So, -15 has a higher priority than 15. The default nice value is 0. Regular users can set lower priorities (positive nice values).To use higher priorities (negative nice values), administrator privileges are required.

We can change the priority of a job that is already running using renice. For example:

**renice 15 -p 196**

This changes the nice value of the job with process id 196 to 15. In this case, no dash is used for the command option when specifying the nice value. The following command changes the nice value of process 196 to -9:

**renice -9 -p 196**

## 1.5 TERMINATION OF JOBS (kill COMMAND)

Programs and processes can be launched in multiple different ways. The following steps will attempt to terminate a process that was launched from a command shell.

The methods to terminate a process that is not attached to a shell or terminal is dealt with later in the post.

**Control - c**

If the program was started from a command shell and the process is still attached to the shell, then you might be able to terminate the process by doing a *control-c* on the command shell.

The Control-C sends a SIGINT or an interrupt signal to the currently active process. Most programs will shutdown but it is also possible for the program to ignore this signal depending on its current state.

If a program is hanging or is in a non-responsive state then Control-C might not be effective in terminating the process.

**Control - \**

When *Control-c* does not work, try issuing *Control-\*. This sends a SIGQUIT or the quit signal to the active process. As in the case of the previous SIGINT signal, it is possible again for the process to ignore this signal.

If again this does not work, then your best bet is to use the ***kill*** command against the process. In order to do that, you need to put the offending process that is currently active into a sleep or suspended mode.

**Control-z and kill**

Issuing a *Control-z* on the shell will suspend the currently running process and return you a prompt. Also, it will print out the job number of the process. You can issue a ***kill*** command against this job number in order to terminate the process. If the job number is 2, then

```
bash$ kill -9 %2
```

where 2 is the job number of the process that is suspended.

If you are unsure of the job number, then you can see all the running jobs in the shell by using the ***jobs*** command. The number in the square brackets is the job number.

```
bash$ jobs

[2]- Running gedit &
```

**Background and kill**

If the above ***kill*** of the suspended process does not terminate the process, then you might have to put the process in the background and try *kill* again. Sometimes a suspended process might be holding on or waiting for device or resource which might prohibit it from terminating.

In order to put the suspended process into an active state, you can issue the **bg** command. This will start running the last suspended process (using *Control-z*) as a background process.

```
bash$ bg
```

Once this is an active process, you can issue a ***kill*** command against the process …

```
bash$ kill -9 $!
```

The **$!** is a special environment variable that holds the process id (pid) of the last background-ed process in the shell. You can also use the exact process id of the process.

In some cases, it is quite possible that the process resists being put into the background or that the shell itself is frozen for some reason. In such cases, you will need another shell as well as the process id of the process in order to terminate the process.

This is also the case when the process was not launched from a command shell and hence is not attached to a command shell.

**Kill with Process id**

If you know the *process id* or *pid* of the process, then issue a ***kill*** command against the process.

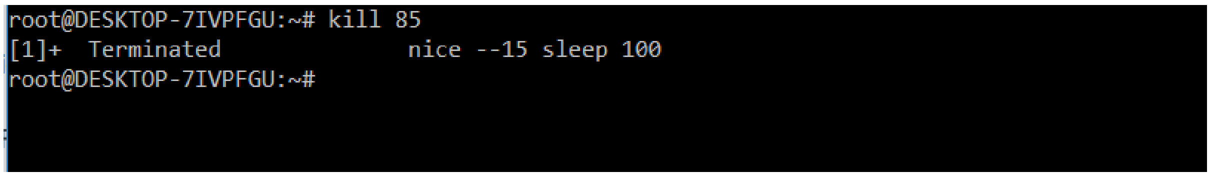
```
bash$ kill -9 9876
```

**Syntax :**

```
kill job_number
```

**Example :**

To kill a background process



```
root@DESKTOP-7IVPFGU:~# kill 85
[1]+  Terminated              nice --15 sleep 100
root@DESKTOP-7IVPFGU:~#
```

```
kill 85
```

It kills the process 85

## 1.6 SCHEDULING JOBS AND RUNNING JOBS IN PERIODICALLY

### at and batch:

At command is used for scheduling the job, but we can set job run only once.

The “at” daemon can be used to run a command or script of your choice. From the command line, you can run the “at” time command to start a job to be run at a specified time. That time can be now; in a specified number of minutes, hours, or days; or at the time of your choice.

To schedule a one-time job at a specific time, type the command at time, where time is the time to execute the command.

The Linux at command argument time can be one of the following:

HH:MM format — For example, 04:00 specifies 4:00AM. If the time is already past, it is executed at the specified time the next day.

- midnight — Specifies 12:00AM.
- noon — Specifies 12:00PM.
- teatime — Specifies 4:00PM.
- month-name day year format — For example, January 15 2002 specifies the 15th day of January in the year 2002. The year is optional.

- MMDDYY, MM/DD/YY, or MM.DD.YY formats — For example, 011502 for the 15th day of January in the year 2002.
- now + time — time is in minutes, hours, days, or weeks. For example, now + 5 days specifies that the command should be executed at the same time in five days.

**Linux at command examples**

| **Command Example** | **Description** |
|---|---|
| at now + 10 minutes | Associated jobs will start in 10 minutes. |
| at now + 2 hours | Associated jobs will start in 2 hours. |
| at now + 1 day | Associated jobs will start in 1 day (24 hours). |
| at now + 1 week | Associated jobs will start in 7 days. |
| at teatime | Associated jobs will start at 4:00 P.M. |
| at 3:00 6/13/07 | Associated jobs will start on June 13, 2007, at 3:00 A.M. |

**Syntax :**

```
at [ option ] [ job ]
```

**Features:**

1. One-off job schedulers
2. 'at' runs based on time schedule
3. 'batch' runs based on system-utilization stats

**Tasks:**

1. Use 'at' to run jobs

a. 'at 15:58'

b. 'at -f at.job.1 16:02'

c. 'at now + 1 day' - runs job 1-day from now (time submitted to job-queue)

**Linux Batch Command:**

Use 'batch' to run jobs

a. 'batch' - supply instructions on STDIN

Note: 'batch' accepts no command line options

Note: 'at' runs the jobs on behalf of 'batch'

Note: 'batch' is simply a special invocation of 'at

## Cron

The Cron service is a time-based job scheduling service that is typically started when the system boots. It checks every minute for any scheduled jobs and runs them if they exist.

## Crontab

In order to manipulate the job schedules, we use the Crontab program in Linux. Crontab, short for '*cron table,*' is a configuration file. Each line of the Crontab represents a job and contains information on what to run and when to run. The following is the format for the Linux Crontab:

M H DOM MON DOW Command

The command gets executed whenever all the time specification fields match the current date and time. More often than not, we use the asterisk (*) symbol in the time specification field to match any value in that field.

Opening and Editing Crontab

Now, let us learn how to open and edit the Crontab file, which is an important step in Linux job scheduling. In order to do so, we use the following command:

crontab -e

After executing this command, you might be prompted to choose an editor. If you are a beginner, I would advise you to select Nano, otherwise feel free to go with any editor that you are comfortable with.

After you are done selecting the editor, a file similar to the one shown in the screenshot below will open up:

Note: The Hash (#) Symbol is used to denote comments. These comments will be ignored by Cron.

Scheduling Jobs

Now in order to schedule jobs in Linux, all you need to do is enter all the necessary details while following the format mentioned above. Here is an example: Suppose I want to run the command *usr/bin/backup* at *2:30 AM* on the first day of every month, then I will add the following line to the Crontab:

30 02 1 * * /usr/bin/backup

30 : 30th Minute

02 : 2 AM

1 : 1st Day

* : Every Month

* : Every Day of the Week

Specifying Multiple Value and Ranges

It is also possible to schedule jobs in Linux to occur at multiple times. Just use a comma (,) to separate the required values. As an example, lets again consider the previous example. Now if i wish to execute the command at *2:30 PM* as well, all i need to do is this:

30 02,14 1 * * /usr/bin/backup

30 : 30th Minute

02,14 : 2 AM and 2 PM

1 : 1st Day

* : Every Month

* : Every Day of the Week

Note: We have specified *2 PM* by *14* , as Crontab utilizes the 24-hour time format

What's more, its even possible to specify a range of time within Crontab in Linux job scheduling. Just insert the values separated by a dash (–). Here, we will again consider our initial example for showing how it's done. This time, suppose we wish to execute the command at every hour between *2 AM* and *2 PM* on the first day of every month. We will do this in the following fashion:

00 02-14 1 * * /usr/bin/backup

00 : 0th Minute

02-14 : 2AM, 3AM, 4AM, 5AM, 6AM, 7AM, 8AM, 9AM, 10AM, 11AM, 12AM, 1PM, 2PM

1 : 1st Day

* : Every Month

* : Every Day of the Week

Crontab Shortcuts

As a bonus in this article on how to schedule jobs in Linux, consider the following shortcuts and keywords of the Crontab format which might come in handy to schedule jobs in Linux:

@yearly : run once a year at midnight on the morning of January 1

@annually : *same as @yearly*

@monthly : run once a month at midnight on the morning of the first day of the month

@weekly : run once a week at midnight on the morning of Sunday

@daily : run everyday at midnight

@midnight : same as @daily

@hourly : run once a hour at the beginning of the hour

For example: Consider that I have to run the */usr/bin/backup* command every month. The Crontab format for it will be:

```
@monthly /usr/bin/backup
```

Once you save the changes to the Crontab and exit, you shall see the following message indicating that you carried out the process successfully.

## 1.7WAIT AND SLEEP COMMANDS

**wait :**

**wait** is a built-in shell command which waits for a given process to complete, and returns its exit status.

If an ID is not given, **wait** waits for all currently active child processes, and the return status is **zero**. If the ID is a job specification, **wait** waits for all processes in the job's pipeline.

**Syntax :**

```
wait [pid] [jobid]
```

**Example :**

```
wait 1384
```

It will wait for process 1384 to finish and then returns exit status.

## sleep :

Sometimes we need to stop or make any process to add delay for some amount of time then sleep is used.

**Syntax:**

```
sleep NUMBER[suffix]
```

Here suffix is one of the following

- **s** for seconds(default)
- **m** for minute
- **h** for hour
- **d** for day

**Example :**

```
sleep 10s
```

Add delay for 10 seconds

```
sleep 10m
```

Add delay for 10 minutes

```
sleep 10h
```

Add delay for 10 hour

## 1.8 LET US SUM UP

**Points to ponder**

- ps command is for checking status of running process or jobs
- We can background or foreground running process
- Normally all process have same priority unless we specify it using nice command
- Sometimes terminating jobs become necessary when they hogs CPU time
- Scheduling tasks and running them periodically by at, batch, cron, crontab commands

## 1.9 FURTHER READING

1. Here is an article about the various Linux commands: https://www.geeksforgeeks.org/linux-commands/
2. Linux/Unix commands tutorial https://www.javatpoint.com/linux-tutorial

## 1.10 ASSIGNMENTS

5) Discuss in detail, the working of the cron and crontab command for running the jobs periodically.
6) Explain the job execution with priority using nice command.

## 1.11 ACTIVITIES

1. Make a process that wishes happy birthday and make it automatically run on your birthday.
2. Try to give process different priorities by nice command and observe their behaviour in different cases.
3. What are various IDs associated with a process ?
4. What are different process states in Linux ?
5. What is Daemon process ?

6. State advantages of running process in Background.

## 1.12 CASE STUDIES

2) Prepare a detail chart of various process management Linux commands as a quick reference manual.
3) Notice process status of processes and make hierarchy of process which are descendant of particular parent process.

# Unit 2: Patterns Finding and Linux Filter Commands

2

## Unit Structure

2.1. Learning Objectives

2.2. Introduction

2.3. Regular Expression and Finding Patterns

2.4. Linux Filter Commands

2.5. Mount and Unmount File System

2.6. Let Us Sum Up

2.7. Further Reading

2.8. Assignments

2.9. Activities

2.10. Case studies

## 2.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Understand and interpret Regular Expression
- Find the patterns in data
- Apply various filters and transform plain text into meaningful data
- Mount and Unmount file system

## 2.2 INTRODUCTION

In this chapter we will learn about Regular Expression and try to or search different patterns or word in files by using grep and other related commands. Then we will discuss different Linux filter commands which are used for manipulating output stream and add meaning to data by customizing them by our needs.

## 2.3 REGULAR EXPRESSION AND FINDING PATTERNS

**What is a Regular expression?**

A regular expression is a concept of matching a pattern in a given string.

**Which commands/programming languages support regular expressions?**

vi, tr, rename, grep, sed, awk, perl, python etc.

**Basic regular expressions:** This set includes very basic set of regular expressions which do not require any options to execute. This set of regular expressions are developed long time back.

^ –Caret/Power symbol to match a starting at the beginning of line.

$ –To match end of the line

* –0 or more occurrence of the previous character.

. –To match any character

[ ] –Range of character

[^char] –negate of occurrence of a character set

<word> –Actual word finding

/–Escape character

Let’s start with our Regexp with examples, so that we can understand it better.

## 1) grep:

grep command will search for line that matches the the specified pattern.

The grep tool has the following options to use regular expressions:

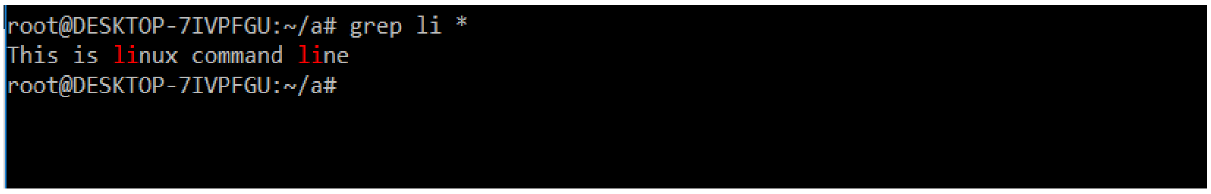
- -E : String is read as ERE (Extended Regular Expressions)
- -G : String is read as BRE (Basic Regular Expressions)
- -P : String is read as PRCE (Perl Regular Expressions)
- -F : String is read literally.
- -R : recursively search for patterns

**Syntax :**

**grep<pattern><fileName>**

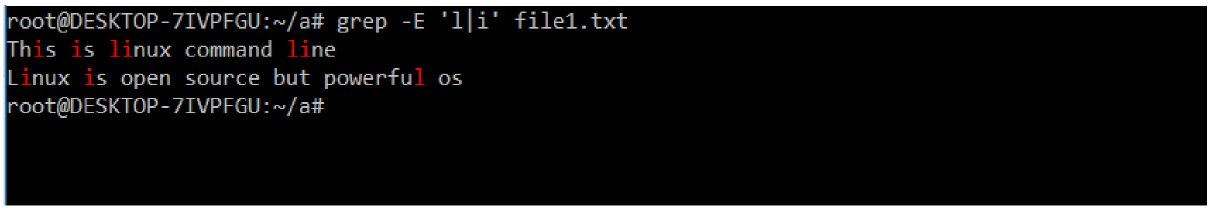
**Example :**

**grep li ***



```
root@DESKTOP-7IVPFGU:~/a# grep li *
This is linux command line
root@DESKTOP-7IVPFGU:~/a#
```

Here * means all the files so it will find li in the all files in directory

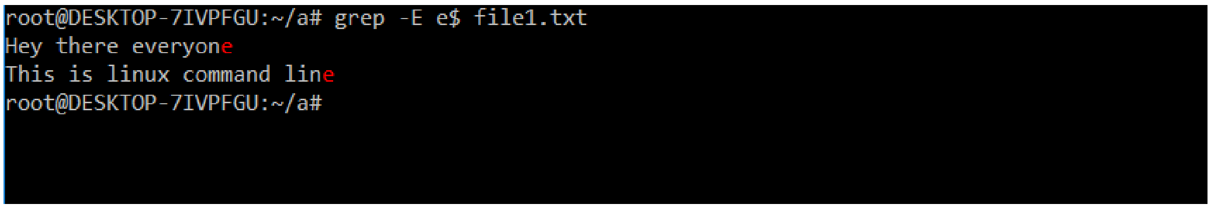


```
root@DESKTOP-7IVPFGU:~/a# grep -E 'l|i' file1.txt
This is linux command line
Linux is open source but powerful os
root@DESKTOP-7IVPFGU:~/a#
```

Here | indicates the or so it will find the line which contains l or i

If we want to find lines that and with specific character then

**grep -E e$ file1.txt**



```
root@DESKTOP-7IVPFGU:~/a# grep -E e$ file1.txt
Hey there everyone
This is linux command line
root@DESKTOP-7IVPFGU:~/a#
```

Hence it will find out the lines which ends with character ‘e’

## 2) egrep :

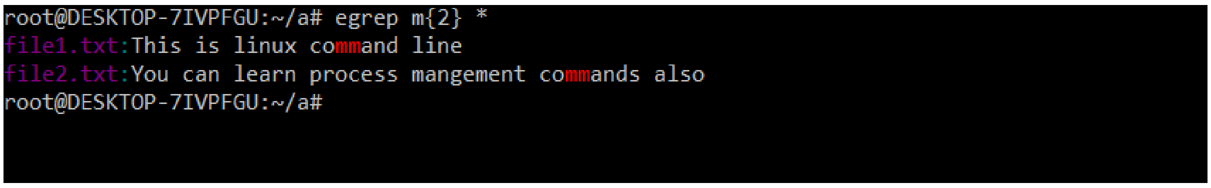
egrep is extended version of grep or egrep is equal to grep -E. Egrep supports more regular expression patterns.

**Syntax :**

`egrep <pattern><fileName>`

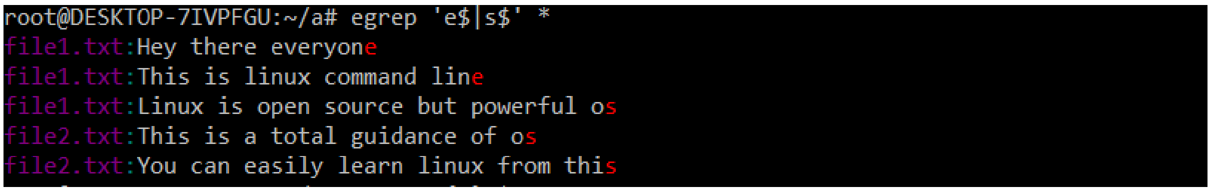
**Example :**

Lets search for lines which contains exactly two consecutive m characters:



```
root@DESKTOP-7IVPFGU:~/a# egrep m{2} *
file1.txt:This is linux command line
file2.txt:You can learn process mangement commands also
root@DESKTOP-7IVPFGU:~/a#
```

Let’s get an output of egrep command with all lines which ends with "e" OR "s":



```
root@DESKTOP-7IVPFGU:~/a# egrep 'e$|s$' *
file1.txt:Hey there everyone
file1.txt:This is linux command line
file1.txt:Linux is open source but powerful os
file2.txt:This is a total guidance of os
file2.txt:You can easily learn linux from this
```

## 3) fgrep :

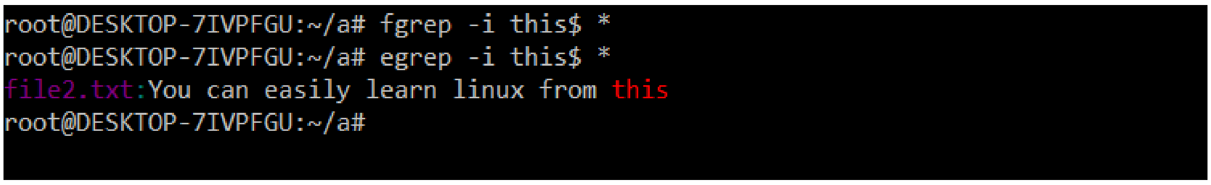
fgrep is a faster version of grep which does not support regular expressions and therefore is considered to be faster. fgrep is equal to grep -F.

**Syntax :**

`fgrep <pattern><fileName>`

**Example :**

Simple proof that fgrep does not interpret regular expressions (regex):



```
root@DESKTOP-7IVPFGU:~/a# fgrep -i this$ *
root@DESKTOP-7IVPFGU:~/a# egrep -i this$ *
file2.txt:You can easily learn linux from this
root@DESKTOP-7IVPFGU:~/a#
```

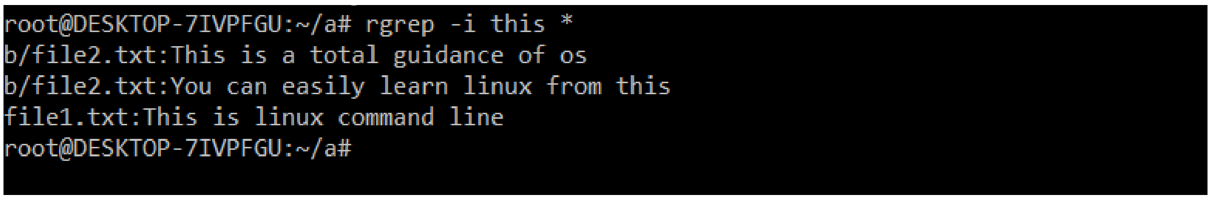
## 4) rgrep :

rgrep is a recursive version of grep. Recursive in this case means that rgrep can recursively descend through directories as it greps for the specified pattern. rgrep is similar to grep -r.

**Syntax :**

**rgrep <pattern><fileName>**

**Example :**



```
root@DESKTOP-7IVPFGU:~/a# rgrep -i this *
b/file2.txt:This is a total guidance of os
b/file2.txt:You can easily learn linux from this
file1.txt:This is linux command line
root@DESKTOP-7IVPFGU:~/a#
```

Here rgrep is showing results from sub directories also see file1.txt is in same directory and file2.txt is in directory b.

## 2.4LINUX FILTER COMMAND

Filters are the programs that take textual data as an input and format it according to the applied filter. These filters have various command line options which provides more flexibility to the output given by the filters. There are number of filters in Linux environment some of them are the following.

A filter, in the context of the Linux command line, is a program that accepts textual data and then transforms it in a particular way. Filters are a way to take raw data, either produced by another program, or stored in a file, and manipulate it to be displayed in a way more suited to what we are after.

### 1) cut:

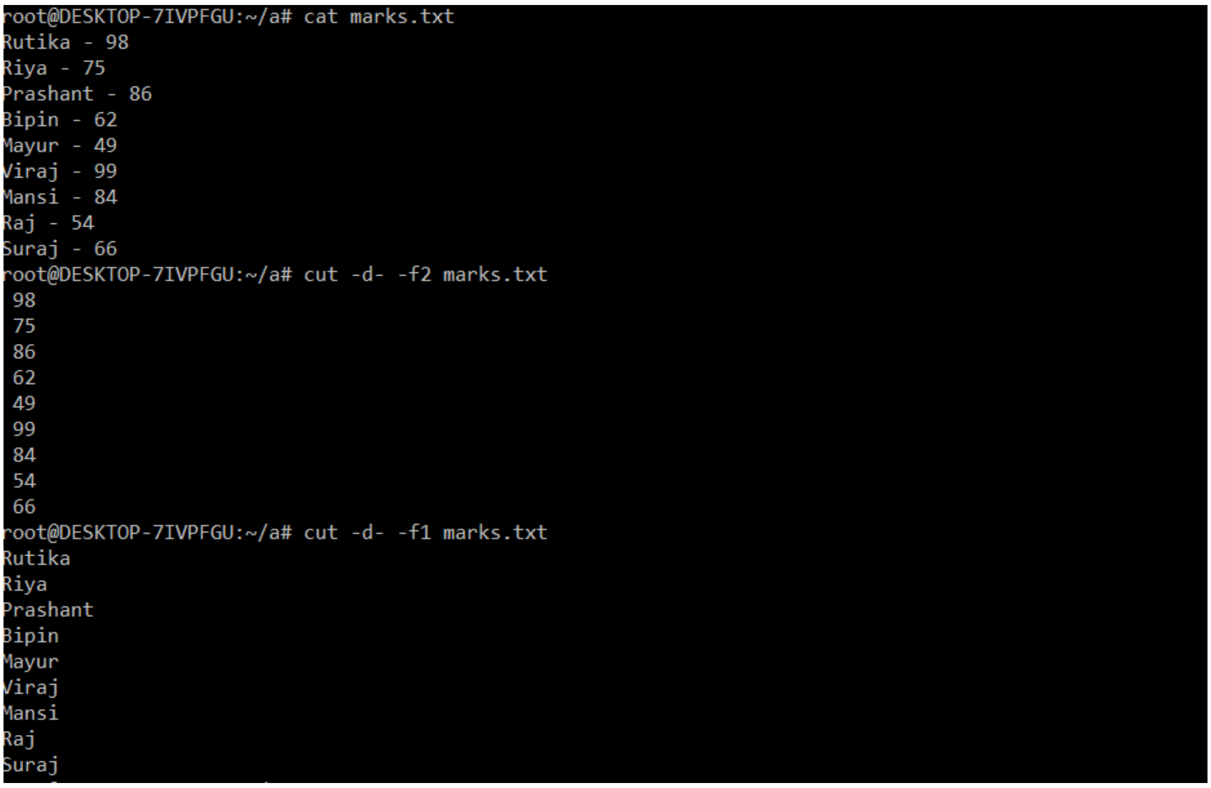
The cut command in Linux is a command line utility for cutting sections from each line of files and writing the result to standard output. It can be used to cut parts of a line by byte position, character and delimiter. It can also be used to cut data from file formats like CSV.

**Syntax :**

**cut [option] [fileName]**

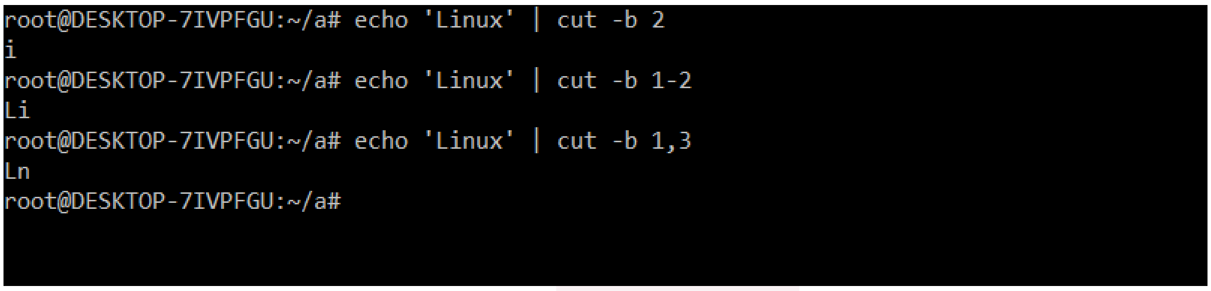
**Example :**

After (-d), delimiter (from where you want to separate the columns) comes. Delimiters can be a space (' '), a hyphen (-), a slash (/) or anything else. After (-f), column number is mentioned.



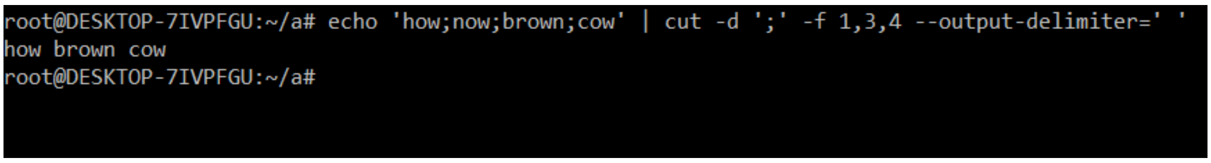
```
root@DESKTOP-7IVPFGU:~/a# cat marks.txt
Rutika - 98
Riya - 75
Prashant - 86
Bipin - 62
Mayur - 49
Viraj - 99
Mansi - 84
Raj - 54
Suraj - 66
root@DESKTOP-7IVPFGU:~/a# cut -d- -f2 marks.txt
 98
 75
 86
 62
 49
 99
 84
 54
 66
root@DESKTOP-7IVPFGU:~/a# cut -d- -f1 marks.txt
Rutika
Riya
Prashant
Bipin
Mayur
Viraj
Mansi
Raj
Suraj
```

To cut out a section of a line by specifying a byte position use the -b option.



```
root@DESKTOP-7IVPFGU:~/a# echo 'Linux' | cut -b 2
i
root@DESKTOP-7IVPFGU:~/a# echo 'Linux' | cut -b 1-2
Li
root@DESKTOP-7IVPFGU:~/a# echo 'Linux' | cut -b 1,3
Ln
root@DESKTOP-7IVPFGU:~/a#
```

To modify the output delimiter use the --output-delimiter option. Note that this option is not available on the BSD version of cut. In the following example a semicolon is converted to a space and the first, third and fourth fields are selected.



```
root@DESKTOP-7IVPFGU:~/a# echo 'how;now;brown;cow' | cut -d ';' -f 1,3,4 --output-delimiter=' '
how brown cow
root@DESKTOP-7IVPFGU:~/a#
```

## 2) pest :

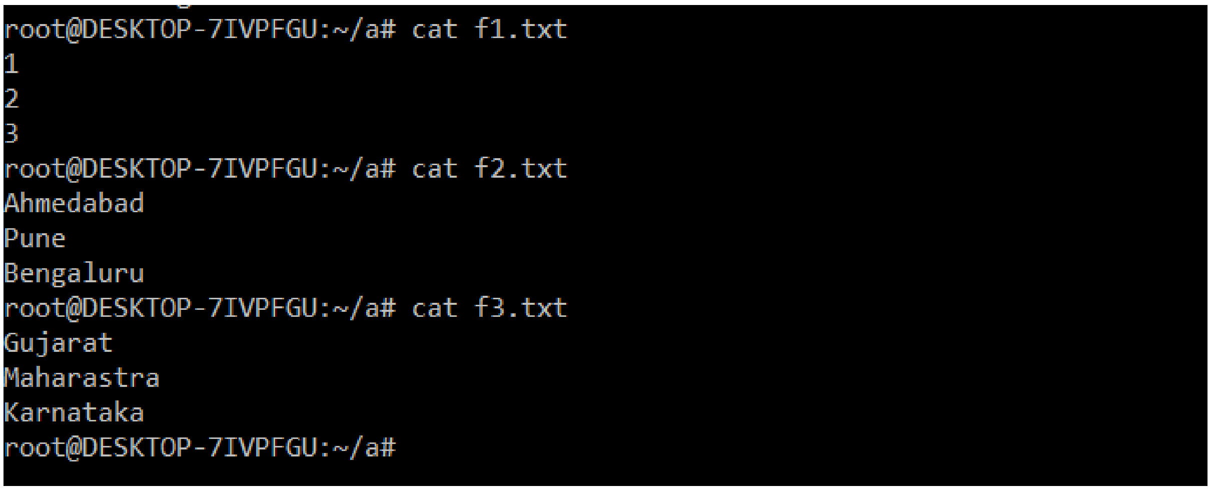
The Linux command is utility which merges lines of the files

**Syntax :**

**Paste [options] [files]**

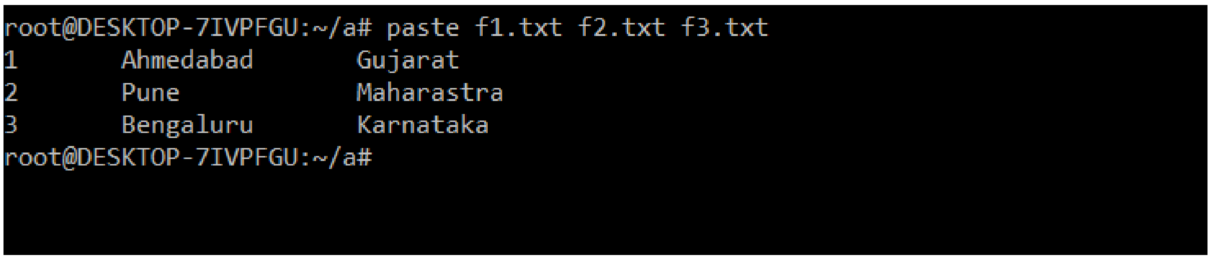
**Example:**

Suppose we have three files - file1.txt, file2.txt, and file3.txt - with following contents:



```
root@DESKTOP-7IVPFGU:~/a# cat f1.txt
1
2
3
root@DESKTOP-7IVPFGU:~/a# cat f2.txt
Ahmedabad
Pune
Bengaluru
root@DESKTOP-7IVPFGU:~/a# cat f3.txt
Gujarat
Maharastra
Karnataka
root@DESKTOP-7IVPFGU:~/a#
```

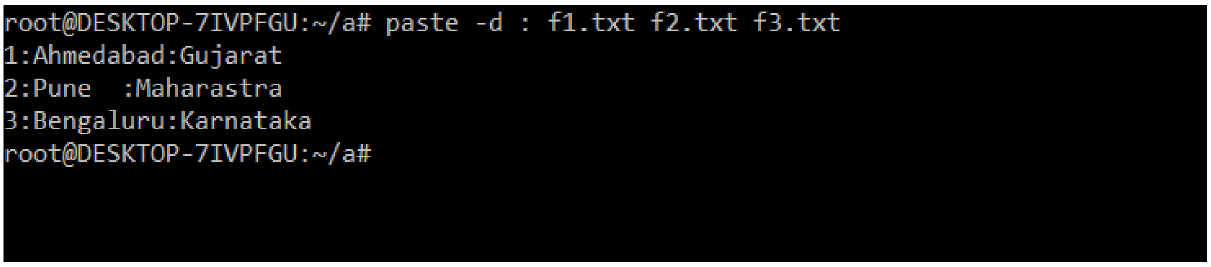
And the task is to merge lines of these files in a way that each row of the final output contains index, country, and continent, then you can do that using paste in the following way:



```
root@DESKTOP-7IVPFGU:~/a# paste f1.txt f2.txt f3.txt
1       Ahmedabad       Gujarat
2       Pune            Maharastra
3       Bengaluru       Karnataka
root@DESKTOP-7IVPFGU:~/a#
```

Sometimes, there can be a requirement to add a delimiting character between entries of each resulting row. This can be done using the **-d** command line option, which requires you to provide the delimiting character you want to use.

For example, to apply a colon (:) as a delimiting character, use the paste command in the following way:



```
root@DESKTOP-7IVPFGU:~/a# paste -d : f1.txt f2.txt f3.txt
1:Ahmedabad:Gujarat
2:Pune   :Maharastra
3:Bengaluru:Karnataka
root@DESKTOP-7IVPFGU:~/a#
```

## 3) join:

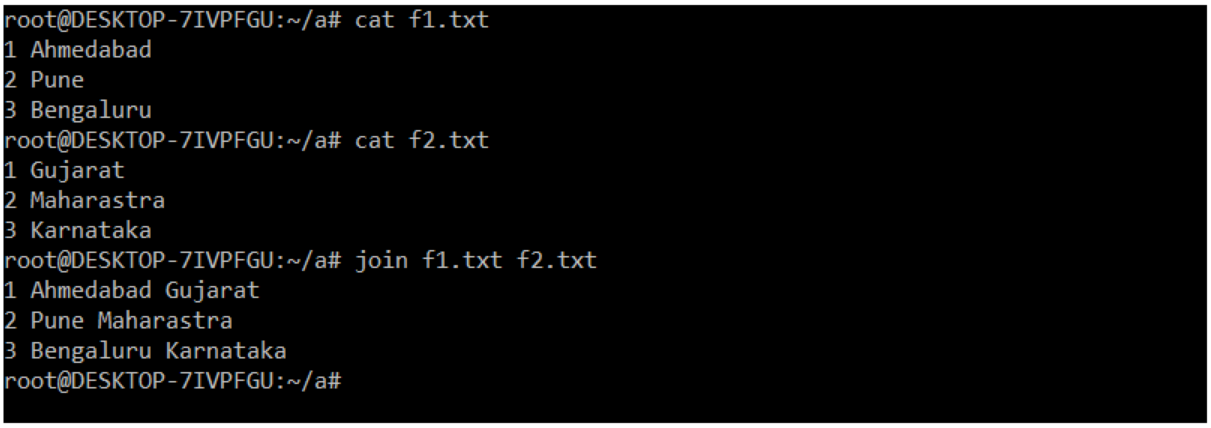
Join command combines two files based on the matching content lines found in each file.

**Syntax:**

**Join [options] FILE1 FILE2**

**Example:**

Consider this example

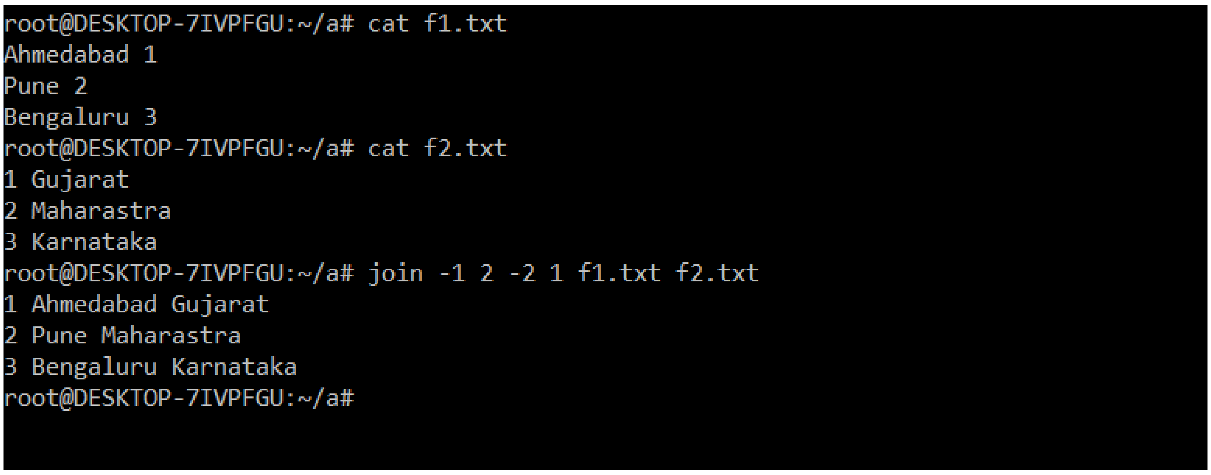


```
root@DESKTOP-7IVPFGU:~/a# cat f1.txt
1 Ahmedabad
2 Pune
3 Bengaluru
root@DESKTOP-7IVPFGU:~/a# cat f2.txt
1 Gujarat
2 Maharastra
3 Karnataka
root@DESKTOP-7IVPFGU:~/a# join f1.txt f2.txt
1 Ahmedabad Gujarat
2 Pune Maharastra
3 Bengaluru Karnataka
root@DESKTOP-7IVPFGU:~/a#
```

Here both of them has common numbering so file is merged in output.

Let’s consider another example in which we will join two files by specific field in both files.

**join -1 2 -2 1 f1.txt f2.txt**



```
root@DESKTOP-7IVPFGU:~/a# cat f1.txt
Ahmedabad 1
Pune 2
Bengaluru 3
root@DESKTOP-7IVPFGU:~/a# cat f2.txt
1 Gujarat
2 Maharastra
3 Karnataka
root@DESKTOP-7IVPFGU:~/a# join -1 2 -2 1 f1.txt f2.txt
1 Ahmedabad Gujarat
2 Pune Maharastra
3 Bengaluru Karnataka
root@DESKTOP-7IVPFGU:~/a#
```

Here -1 2 mean first file’s second field and -2 1 means second file’s 1st field.

## 4) sort :

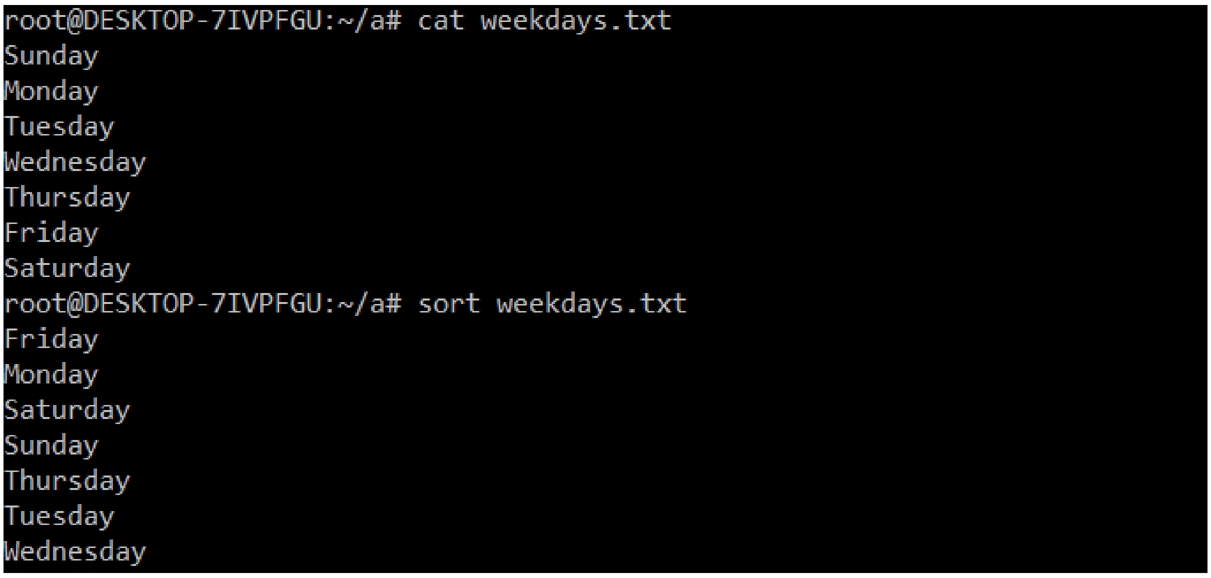
The sort command in Linux sort the file contents in alphabetical order.

**Syntax :**

`sort FILE`

**Example :**

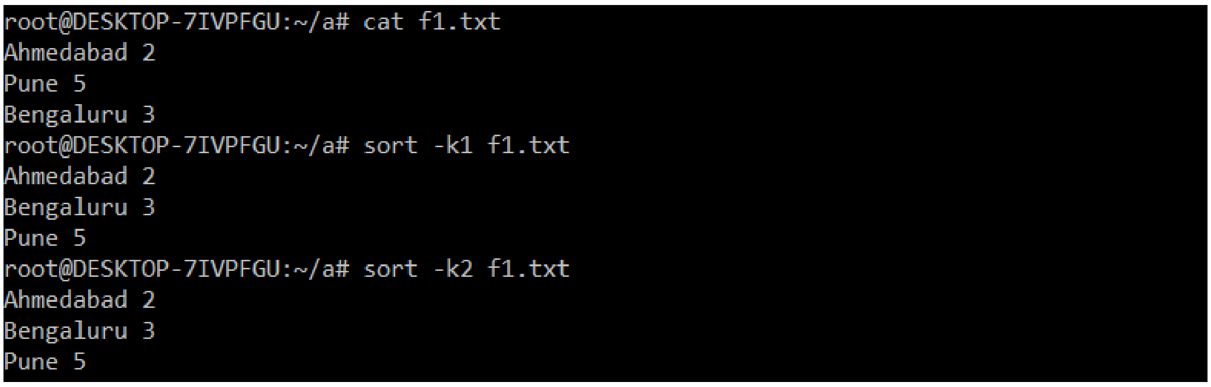
Here in file we have weekdays sort will sort it in alphabetical order



```
root@DESKTOP-7IVPFGU:~/a# cat weekdays.txt
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
root@DESKTOP-7IVPFGU:~/a# sort weekdays.txt
Friday
Monday
Saturday
Sunday
Thursday
Tuesday
Wednesday
```

If file has more than one columns then column no is used to sort file by that column using option **-k**

After -k column number must be specified. See below example:



```
root@DESKTOP-7IVPFGU:~/a# cat f1.txt
Ahmedabad 2
Pune 5
Bengaluru 3
root@DESKTOP-7IVPFGU:~/a# sort -k1 f1.txt
Ahmedabad 2
Bengaluru 3
Pune 5
root@DESKTOP-7IVPFGU:~/a# sort -k2 f1.txt
Ahmedabad 2
Bengaluru 3
Pune 5
```

## 5) uniq:

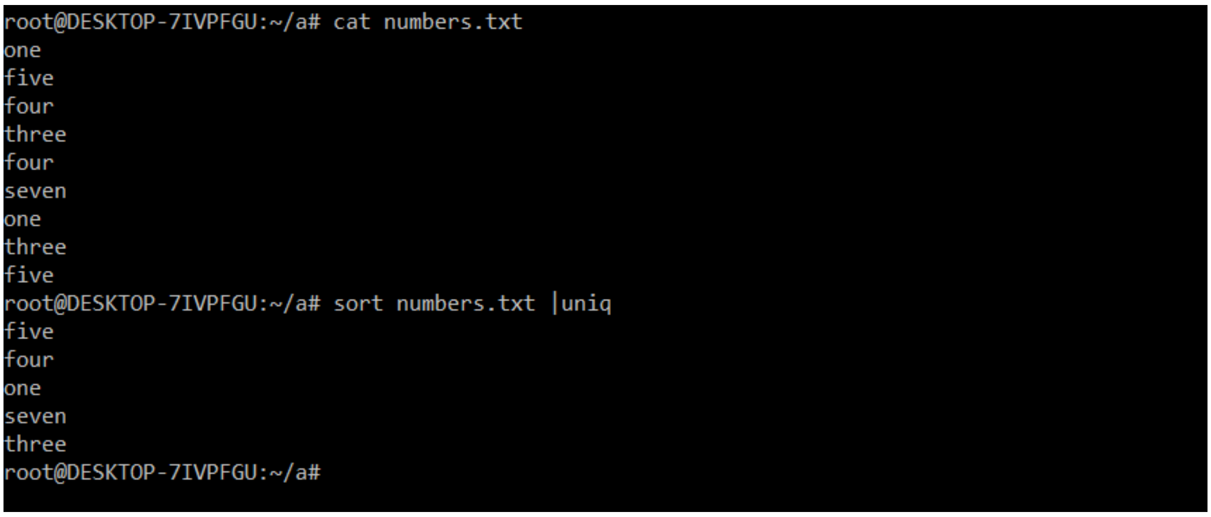
Uniq will report or omits the repeated lines

**Syntax :**

**uniq [option]**

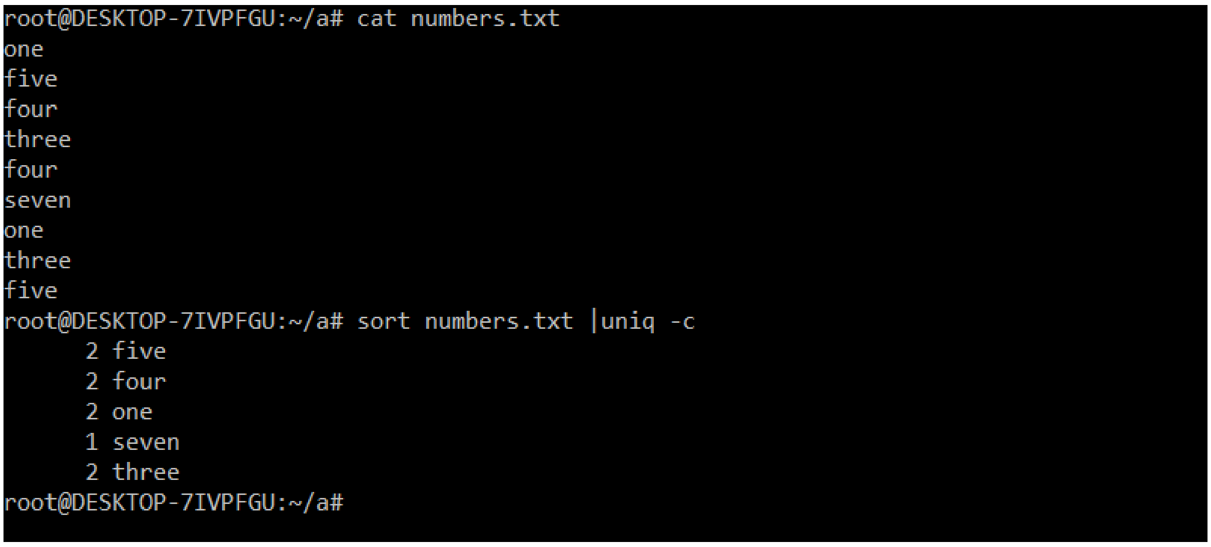
**Example:**

With uniq we can form sorted list in which every word occurs only once. See the example.



```
root@DESKTOP-7IVPFGU:~/a# cat numbers.txt
one
five
four
three
four
seven
one
three
five
root@DESKTOP-7IVPFGU:~/a# sort numbers.txt |uniq
five
four
one
seven
three
root@DESKTOP-7IVPFGU:~/a#
```

We can number of occurance of each line in file by **uniq -c**



```
root@DESKTOP-7IVPFGU:~/a# cat numbers.txt
one
five
four
three
four
seven
one
three
five
root@DESKTOP-7IVPFGU:~/a# sort numbers.txt |uniq -c
      2 five
      2 four
      2 one
      1 seven
      2 three
root@DESKTOP-7IVPFGU:~/a#
```

## 6) comm :

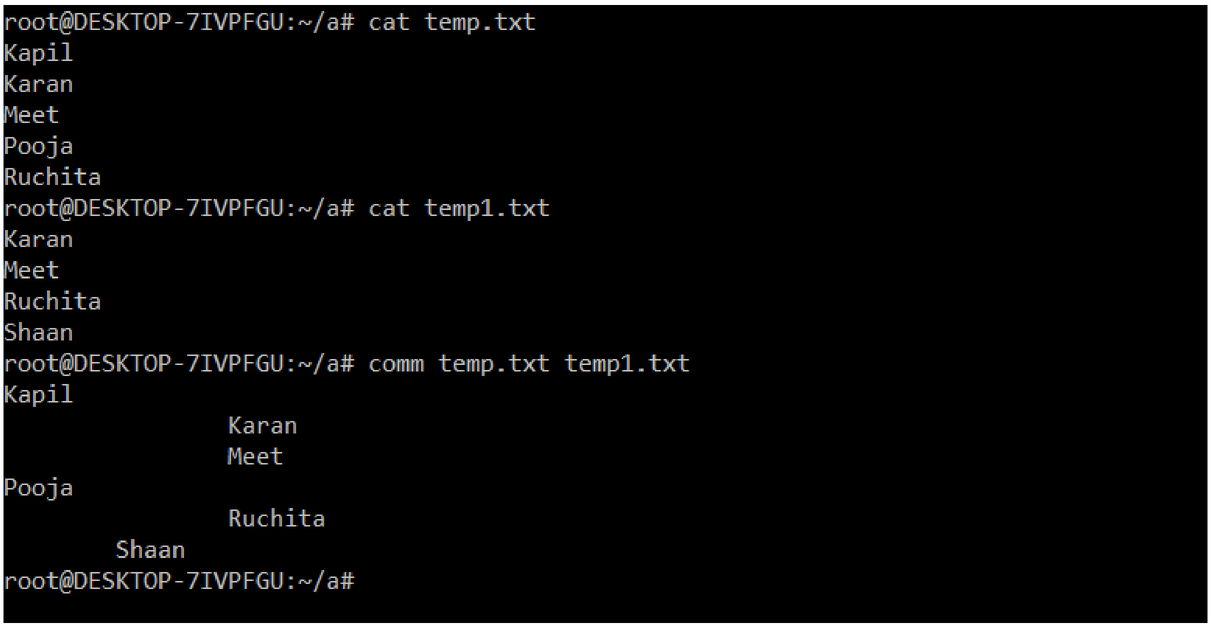
The 'comm' command compares two files or streams. By default, 'comm' will always display **three columns**. First column indicates non-matching items of first file,second column indicates non-matching items of second file, and third column indicates matching items of both the files. Both the files has to be in sorted order for 'comm' command to be executed.

**Syntax:**

**comm File1 File2 [File ..]**

**Example :**

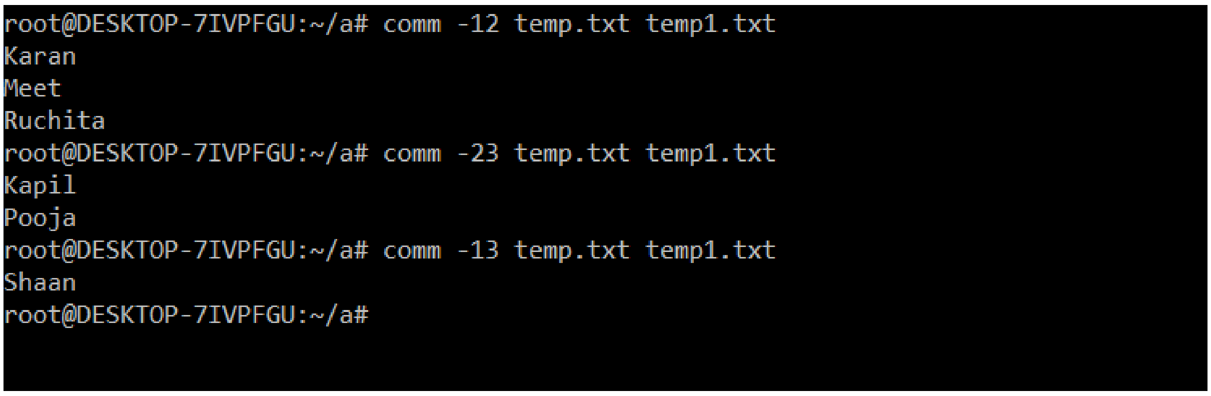
Simple example for comm



```
root@DESKTOP-7IVPFGU:~/a# cat temp.txt
Kapil
Karan
Meet
Pooja
Ruchita
root@DESKTOP-7IVPFGU:~/a# cat temp1.txt
Karan
Meet
Ruchita
Shaan
root@DESKTOP-7IVPFGU:~/a# comm temp.txt temp1.txt
Kapil
                Karan
                Meet
Pooja
                Ruchita
        Shaan
root@DESKTOP-7IVPFGU:~/a#
```

Here third column consist common lines in both file.

To display only one column we need to specify which column must not be display.



```
root@DESKTOP-7IVPFGU:~/a# comm -12 temp.txt temp1.txt
Karan
Meet
Ruchita
root@DESKTOP-7IVPFGU:~/a# comm -23 temp.txt temp1.txt
Kapil
Pooja
root@DESKTOP-7IVPFGU:~/a# comm -13 temp.txt temp1.txt
Shaan
root@DESKTOP-7IVPFGU:~/a#
```

## 7) cmp :

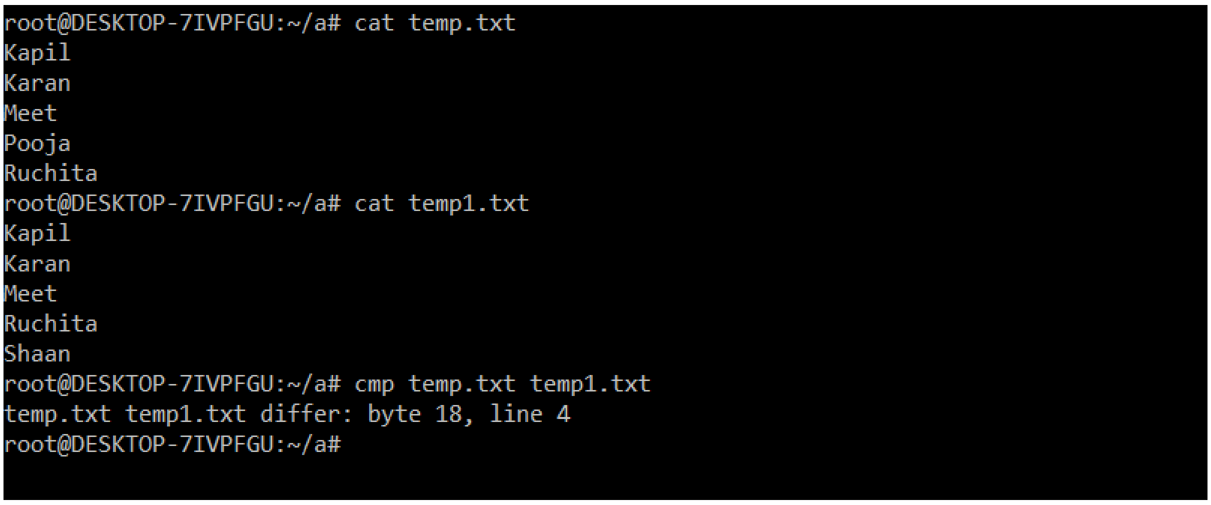
The cmp utility compares two files of any type and writes the results to the standard output. By default, cmp is silent if the files are the same; if they differ, the byte and line number at which the first difference occurred is reported.

Bytes and lines are numbered beginning with one.

**Syntax :**

**cmp FILE1 [FILE2 [SKIP] ]**

**Example :**



```
root@DESKTOP-7IVPFGU:~/a# cat temp.txt
Kapil
Karan
Meet
Pooja
Ruchita
root@DESKTOP-7IVPFGU:~/a# cat temp1.txt
Kapil
Karan
Meet
Ruchita
Shaan
root@DESKTOP-7IVPFGU:~/a# cmp temp.txt temp1.txt
temp.txt temp1.txt differ: byte 18, line 4
root@DESKTOP-7IVPFGU:~/a#
```

Here two files differs from line no 4 hence it gives that particular no as output and also no of byte from where it differs.

## 8) sed:

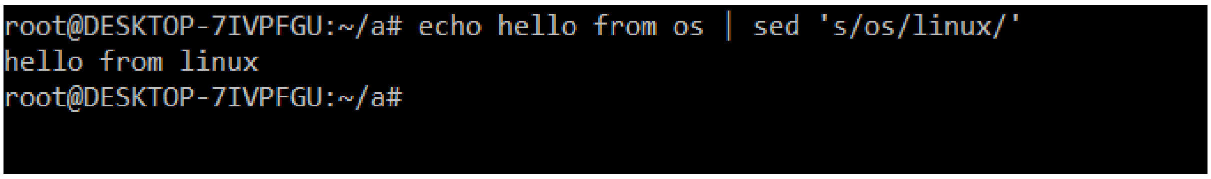
Command 'sed' stands for stream editor. You can use this command to edit streams (files) using regular expressions. But this editing is not permanent. It remains only in display, but in actual file content remains same.

**Syntax :**

**command | sed ‘s/OLD WORD/NEW WORD’**

**Example :**

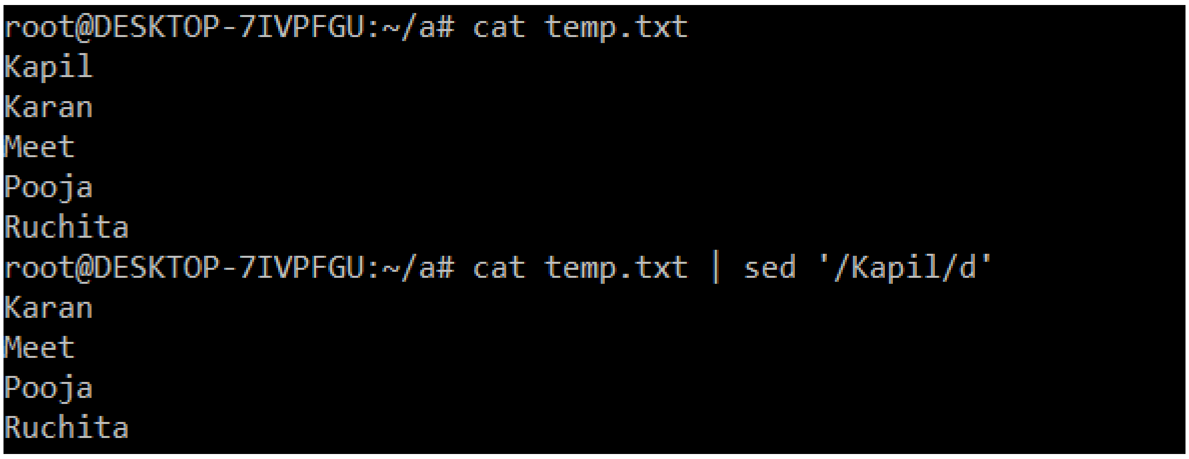
echo hello from os | sed ‘s/os/linux’



```
root@DESKTOP-7IVPFGU:~/a# echo hello from os | sed 's/os/linux/'
hello from linux
root@DESKTOP-7IVPFGU:~/a#
```

This will replace os with linux in output

To remove line we can use **sed ‘/word/d’**



```
root@DESKTOP-7IVPFGU:~/a# cat temp.txt
Kapil
Karan
Meet
Pooja
Ruchita
root@DESKTOP-7IVPFGU:~/a# cat temp.txt | sed '/Kapil/d'
Karan
Meet
Pooja
Ruchita
```

## 9) tr :

The command tr is used to **translate** , like from lowercase to uppercase or delete any word or characters

Here are some different sets

[CHAR*]

in SET2, copies of CHAR until length of SET1

[CHAR*REPEAT]

REPEAT copies of CHAR, REPEAT octal if starting with 0

[:alnum:]

all letters and digits

[:alpha:]

all letters

[:blank:]

all horizontal whitespace

[:cntrl:]

all control characters

[:digit:]

all digits

[:graph:]

all printable characters, not including space

[:lower:]

all lower case letters

[:print:]

all printable characters, including space

[:punct:]

all punctuation characters

[:space:]

all horizontal or vertical whitespace

[:upper:]

all upper case letters

[:xdigit:]

all hexadecimal digits

[=CHAR=]

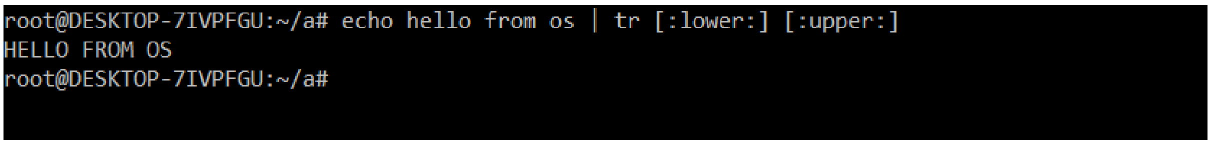
all characters which are equivalent to CHAR

**Syntax :**

**command | tr [option] set1 [set2]**

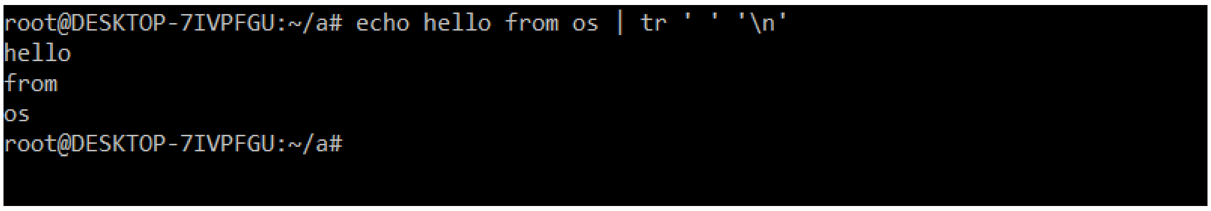
**Example :**

Here we are translating a string from lower to upper



```
root@DESKTOP-7IVPFGU:~/a# echo hello from os | tr [:lower:] [:upper:]
HELLO FROM OS
root@DESKTOP-7IVPFGU:~/a#
```

Consider this following example in that we are replacing all blank spaces by \n(new line)



```
root@DESKTOP-7IVPFGU:~/a# echo hello from os | tr ' ' '\n'
hello
from
os
root@DESKTOP-7IVPFGU:~/a#
```

## 10) od :

The 'od' term stands for octal dump. It displays content of a file in different human-readable formats like hexadecimal, octal and ASCII characters.

**Syntax :**

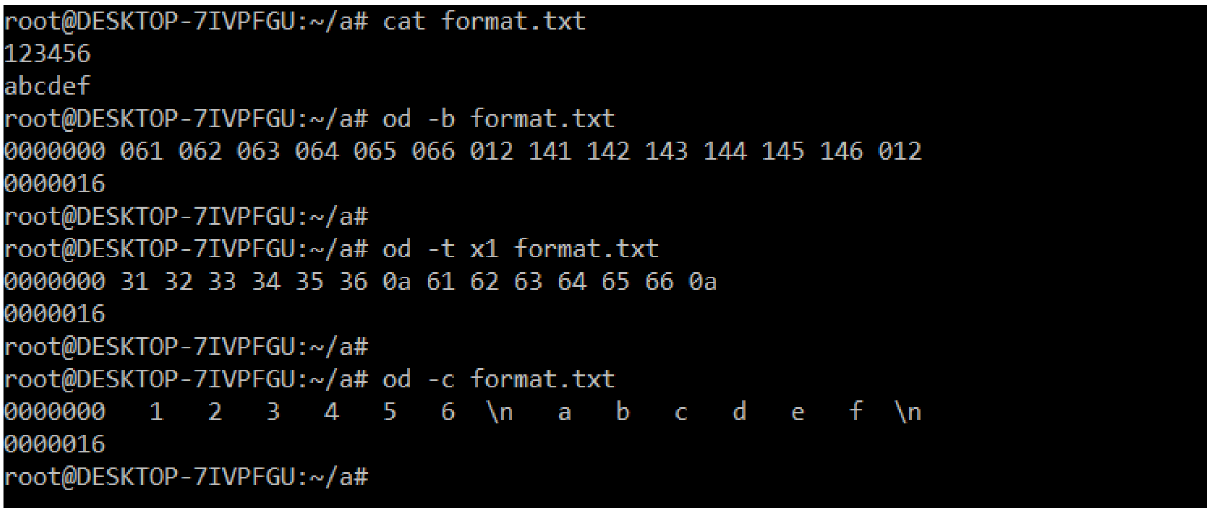
**od [option] fileName**

**Example :**

od -b **<fileName>** (display files in octal format)

od -t x1 **<fileName>** (display files in hexadecimal bytes format)

od -c **<fileName>** (display files in ASCII (back slashed) character format)


```
root@DESKTOP-7IVPFGU:~/a# cat format.txt
123456
abcdef
root@DESKTOP-7IVPFGU:~/a# od -b format.txt
0000000 061 062 063 064 065 066 012 141 142 143 144 145 146 012
0000016
root@DESKTOP-7IVPFGU:~/a#
root@DESKTOP-7IVPFGU:~/a# od -t x1 format.txt
0000000 31 32 33 34 35 36 0a 61 62 63 64 65 66 0a
0000016
root@DESKTOP-7IVPFGU:~/a#
root@DESKTOP-7IVPFGU:~/a# od -c format.txt
0000000   1   2   3   4   5   6  \n   a   b   c   d   e   f  \n
0000016
root@DESKTOP-7IVPFGU:~/a#
```

## 2.5 MOUNT AND UNMOUNT FILESYSTEM

### Mount a file system

The mount command attaches the filesystem of an external device to the filesystem of a system.

It mounts the external storage devices like hard disks, pen drives, USBs etc.

It instructs the operating system that filesystem is ready to use and associate it with a particular point in the system's hierarchy. Mounting will make files, directories and devices available to the users.

Here, this command instructs kernel to attach filesystem of device at the specified directory.

If destination directory is not mentioned, by default, it mounts the device in the /etc/fstab file.

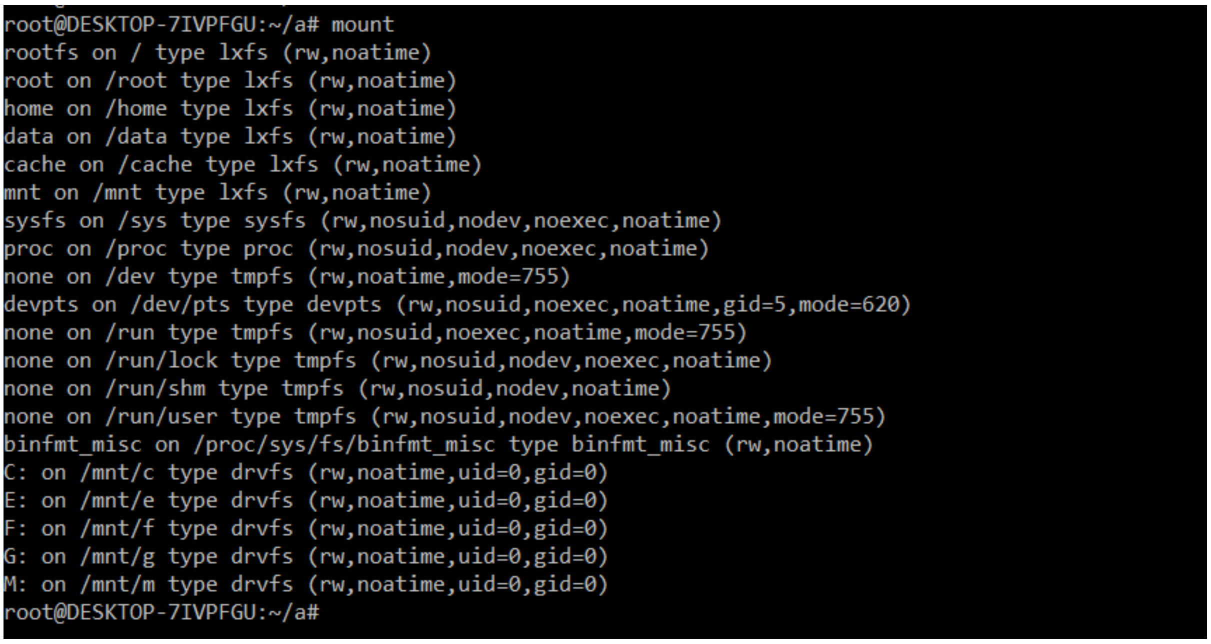
Conversely, **umount**command unmount the mount point and detach the device from the system.

**Syntax :**

```
     mount   [-t   file_system_type]   [-o   mount_options]
device mount_point_directory
```

**Example :**

To view already mounted file system just type mount and hit enter



```
root@DESKTOP-7IVPFGU:~/a# mount
rootfs on / type lxfs (rw,noatime)
root on /root type lxfs (rw,noatime)
home on /home type lxfs (rw,noatime)
data on /data type lxfs (rw,noatime)
cache on /cache type lxfs (rw,noatime)
mnt on /mnt type lxfs (rw,noatime)
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,noatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,noatime)
none on /dev type tmpfs (rw,noatime,mode=755)
devpts on /dev/pts type devpts (rw,nosuid,noexec,noatime,gid=5,mode=620)
none on /run type tmpfs (rw,nosuid,noexec,noatime,mode=755)
none on /run/lock type tmpfs (rw,nosuid,nodev,noexec,noatime)
none on /run/shm type tmpfs (rw,nosuid,nodev,noatime)
none on /run/user type tmpfs (rw,nosuid,nodev,noexec,noatime,mode=755)
binfmt_misc on /proc/sys/fs/binfmt_misc type binfmt_misc (rw,noatime)
C: on /mnt/c type drvfs (rw,noatime,uid=0,gid=0)
E: on /mnt/e type drvfs (rw,noatime,uid=0,gid=0)
F: on /mnt/f type drvfs (rw,noatime,uid=0,gid=0)
G: on /mnt/g type drvfs (rw,noatime,uid=0,gid=0)
M: on /mnt/m type drvfs (rw,noatime,uid=0,gid=0)
root@DESKTOP-7IVPFGU:~/a#
```

By using mount, you can override the default settings in /etc/fstab. For example, entering the following mounts the partition /dev/sdd1 to the directory /data:

```
# mount /dev/sdd1 /data
```

You do not usually specify the file system type because it is recognized automatically (using magic numbers in the superblock, or simply by trying different file system types; see man mount for details).

## Unmount a File System

Once a file system is mounted, you can use the umount command (without an "n") to unmount the file system. You can unmount the file system by using umount with the device or the mount point.

For example to unmount a file system (dev/sdd1) mounted at /data, you could enter one of the following:

```
# umount /data
```

or

```
umount /dev/sdd1
```

In order to unmount the file system, no application or user may use the file system. If it is being used, Linux sees the file system as being "busy" and will refuse to unmount the file system and will produce the below error.

```
# umount /data
umount: /data: target is busy.
```

(In some cases useful info about processes that use the device is found by **lsof(8) or fuser(1)**)

Unmounting is not possible if the mount point is accessed by a process. For umount to be successful, the process needs to stop accessing the mount point.

The **lsof** command lists all open file and processes accessing them in the provided directory. It is useful to identify which processes currently prevent the file system from successful umounting.

```
# lsof /data
COMMAND    PID    USER   FD   TYPE DEVICE  SIZE/OFF     NODE NAME
bash      1160    root  cwd    DIR  253,1       180  4194369 /root
rsyslogd   566    root  cwd    DIR  253,1       224       64 /
...
```

You can also use the fuser command to get the process IDs of the processes currently running on the mpunt point you want to umount.

```
# fuser -cu /data
```

You can also kill all the processes on the mount point using the fuser command.

```
# fuser -ck /data
```

Once the processes are identified, an action can be taken, such as waiting for the process to complete or sending a SIGTERM or SIGKILL signal to the process. In this case, it is sufficient to umount the mount point.

```
# umount /data
```

Note: A common cause for the file system on the mount point to be busy is if the current working directory of a shell prompt is below the active mount point. The process accessing the mount point is bash. Changing to a directory outside the mount point allows the device to be unmounted.

## 2.6 LET US SUM UP

**Points to ponder**

- grep command is used for finding particular word or pattern in file or input string.
- You can apply filters and manipulate and add meaning to your data using Linux filter commands.
- comm command shows common data as well as unique data of both file
- tr command is used to translate like lowercase to upper or delete a character or word
- By sed command we can edit file (streams) using regular expression.
- Sort command sor file data in alphabetical order.

## 2.7 FURTHER READING

1. Here is an article about the various Linux commands: https://www.geeksforgeeks.org/linux-commands/
2. Linux/Unix commands tutorial https://www.javatpoint.com/linux-tutorial

## 2.8 ASSIGNMENTS

1. Discuss various use cases of REGEX and pattern matching.
2. Explain the use of the sed command in Linux.
3. Write a short note on: Linux Filter commands.
4. Unmount and mount filesystem on your pen-drive.

## 2.9 ACTIVITIES

**Q1. Execute the following commands:**

6) Display given file in uppercase only
7) Display full names like January if we enter jan, janu, janua, and so on...
8) Display the number of occurrence of a word in file.

9) Merge and sort data of two file in third file.
10) Mount file system on pen-drive.
11) Find no of unique words in file Set all the access rights to your any one of the directory
12) Display all file having an extension “.txt”

## 2.10 Case Studies

2) Study various filesystems of different OS and try to find out how there are different from each other.

# Unit 3: Inter-Process Communication

3

## Unit Structure

3.10. Learning Objectives

3.11. Introduction

3.12. Commands Related to IPC

3.13. Two-Way Communication

3.14. Write Command

3.15. Mail Command

3.16. Finger Command

3.17. Telnet: Remote Login

3.18. Let Us Sum Up

3.19. Check Your Progress

3.20. Further Reading

3.21. Assignments

3.22. Activities

## 3.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- understand the basics of Inter-process Communication (IPC)
- use the mail, finger, etc. more commands
- do the remote login

## 3.2 INTRODUCTION

This unit discusses about the Inter-process communication. IPCis the method that allows multiple concurrent process to communicate with each other.

Simultaneously concurrently running process can talk to each other such communication known as Inter-process communication.

Basically, there are two types of process i.e., independent and cooperating process. An independent processes are not affected by the execution of other processes while this matters in case of the cooperating processes.

In this communication it could involve a process letting another process know that some event has occurred or the transferring of data from one process to another.

Processes can communicate with each other using the following ways: 1) shared memory 2) message passing

**Following are some important terms that you need to know before proceeding further on this topic.**

**Pipes** – The symbol "|" is known as pipe operator. It take the output from one program and provide piping to other program. A pipe is a method for inter-process communication in which data is taken as input by the pipe from one process and given as output as output to another process. This pipe has no name and so it is known as Unnamed Process. The data send by pipe is stored in buffer.

**Example:**

```
Who | wc –l
```

In this command, first of all who will be processed which will send the information of the currently user logged on the system to the wc –l (word count) command. This command will print the number of lines which indicates the total number of the user who are logged in the system.

Similarly, to count the number of words, characters and lines of any given file we can use cat command along with the wc command.

**Example**

cat temp | wc

**FIFO** – A FIFO is similar to a pipe. A FIFO (First In First Out) is a one-way flow of data. FIFOs have a name, so unrelated processes can share the FIFO. FIFO is a named pipe. This is the main difference between pipes and FIFOs.

Creating A FIFO:

A FIFO is created by the mkfifo function:

Its prototype is given as follows

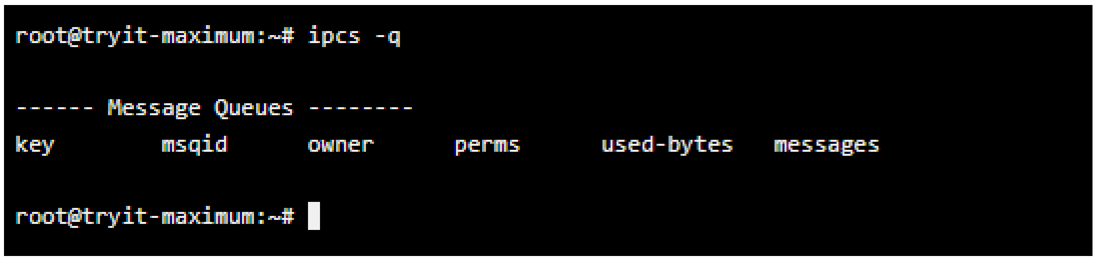
int mkfifo (const char *pathname, mode_tmode);

Where, pathname describe as UNIX pathname (path or filename).

mode – the file permission bits.

**Message Queues** – One process establishes a message queue that others may access. Often a server will establish a message queue so that multiple clients can access. It permit exchange of data between processes.

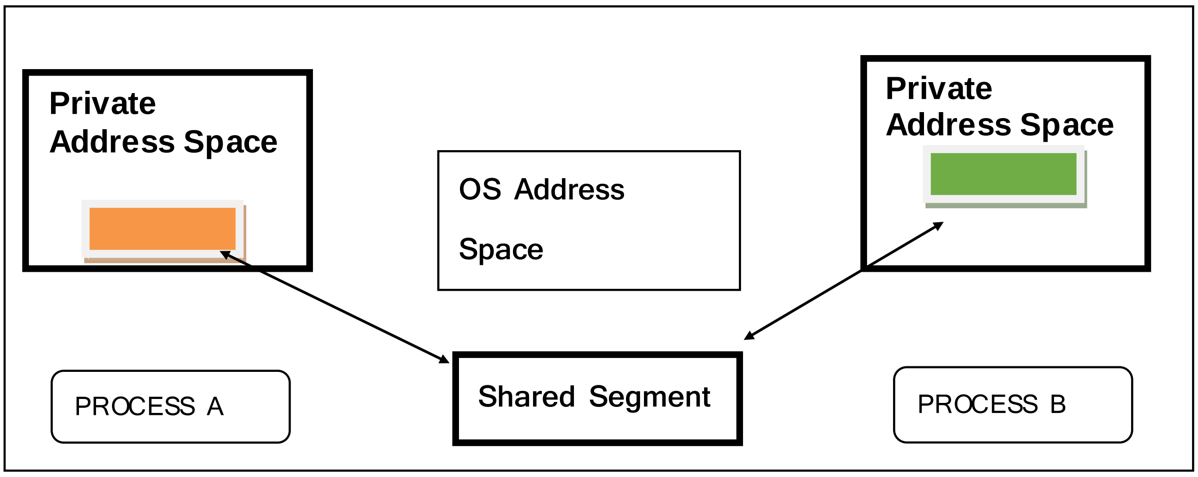
## List all the Message Queue:-



```
root@tryit-maximum:~# ipcs -q

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

root@tryit-maximum:~# 
```

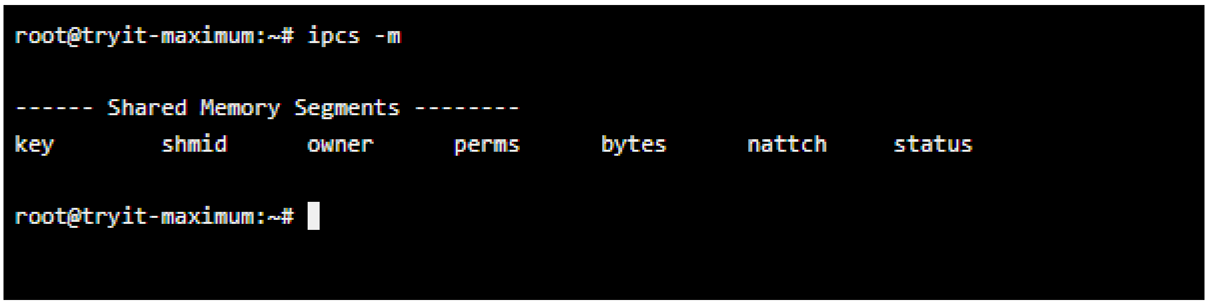
**Here, "-q"** - Option is use to print information about all active message queues.

**Shared Memory**- Shared memory is one of the three inter-process communication (IPC) mechanisms available under Linux and other Unix-like systems. The other two IPC mechanisms are the message queues and semaphores. In case of shared memory, a shared memory segment is created by the kernel and mapped to the data segment of the address space of a requesting process. A process can use the shared memory just like any other global variable in its address space.



As shown in Figure, in this process requesting the segment and operating system maintains the segment. The processes can attached the segment or may de-attached the segment.

## List all the Shared Memory:-

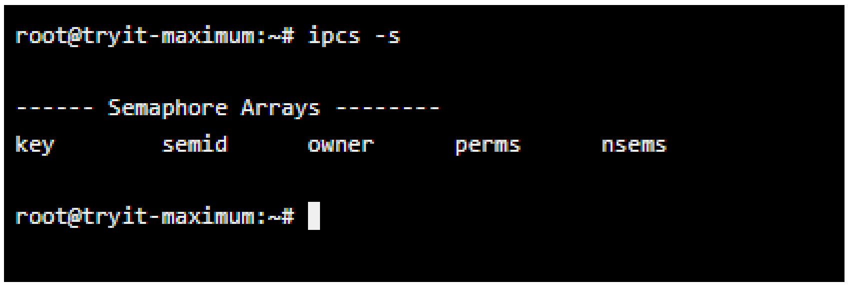


```
root@tryit-maximum:~# ipcs -m

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status

root@tryit-maximum:~# 
```

**Semaphores** –Semaphore is use to describe as a variable. That is used to for controlling the access of a shared resources for multiple processes.

There are two types of semaphores: *binary semaphores* and *counting semaphores*.

**List all the Semaphores:-**



```
root@tryit-maximum:~# ipcs -s

------ Semaphore Arrays --------
key        semid      owner      perms      nsems

root@tryit-maximum:~# 
```

**Here, "-s"** - Option is use to print information about active semaphores set.

a) **Binary semaphore: -**

- Binary semaphores include 2 methods in it.
  - o up , down
  - o lock , unlock
- Binary semaphore grab only 2 values 0 and 1.
- The wait operation only works when the semaphore is 1 and the signal operation succeeds when semaphore is 0.
- It is used to acquire the locks on the process.
- It is easier to implement the binary semaphores than other.

b) **Counting semaphore: -**

- Counting semaphore can have contain value more than two.
- It uses to coordinate the resource access (Number of resource access is available).
- When the resource is added into the semaphore and it is counted automatically increment in semaphore.
- And when resources are removed from semaphore, the count is decremented automatically.

ipcs shows information on the inter-process communication facilities for which the calling process has read access.By default it shows information about all three resources: shared memory segments, message queues, and semaphore arrays. Without options, information shall be written in short format for message queues, shared memory segments, and semaphore sets that are currently active in the system. Otherwise, the information that is displayed is controlled by the options specified.

**Options along with ipcs command which yields following information**

-q: active message queues.

-m: active shared memory segments.

-s: active semaphore sets.

-b: allowable size. (Maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores.)

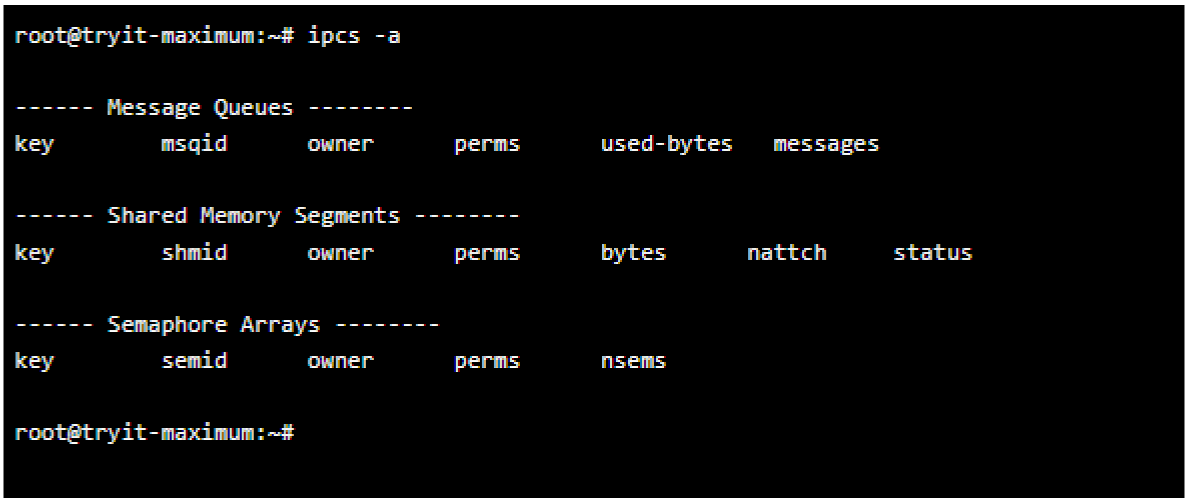
-c: creator's user name and group name.

-p: process number information. (Process ID of the last process to send a message and process ID of the last process to receive a message on message queues, process ID of the creating process, and process ID of the last process to attach or detach on shared memory segments.)

-t: time information.

## 3.3 COMMANDS RELATED TO IPC

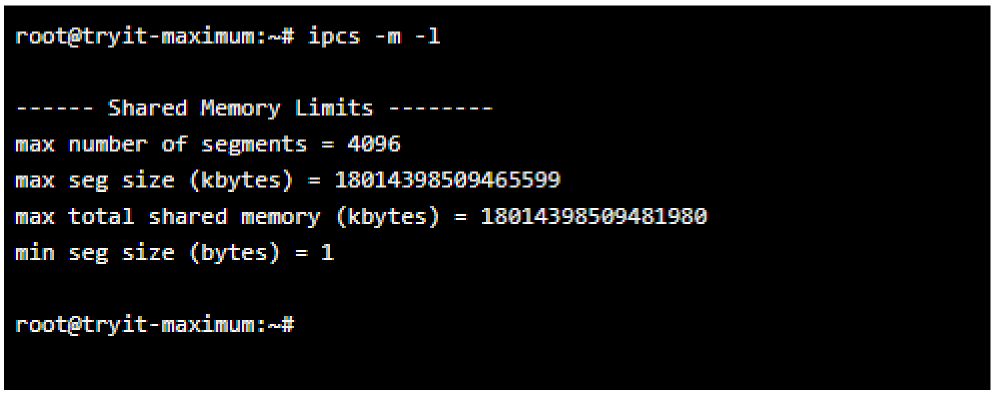
**1) List all the IPC facility: -**



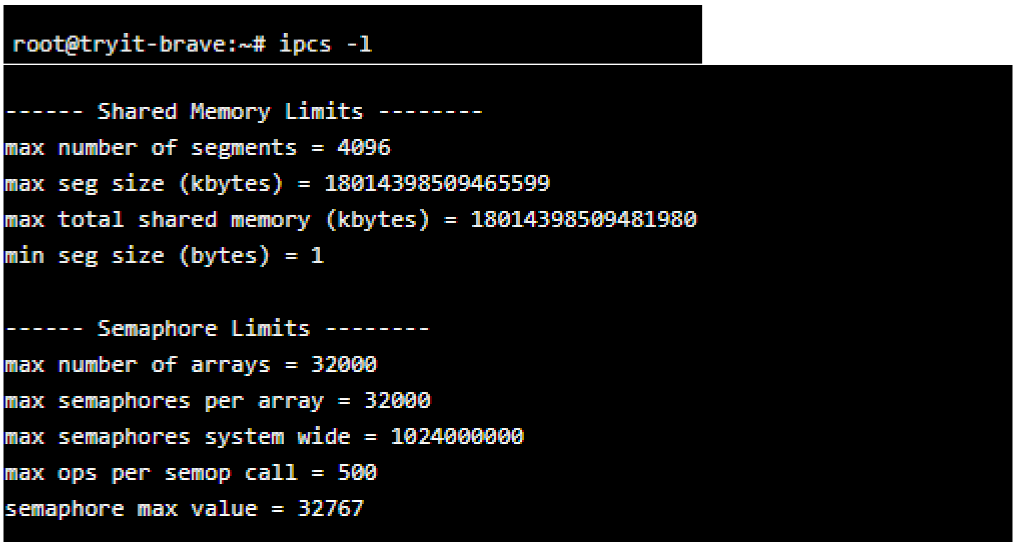
```
root@tryit-maximum:~# ipcs -a

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status

------ Semaphore Arrays --------
key        semid      owner      perms      nsems

root@tryit-maximum:~#
```

**Here, "-a"** - Option is use to print all options.

**2) Display shared memory limits**

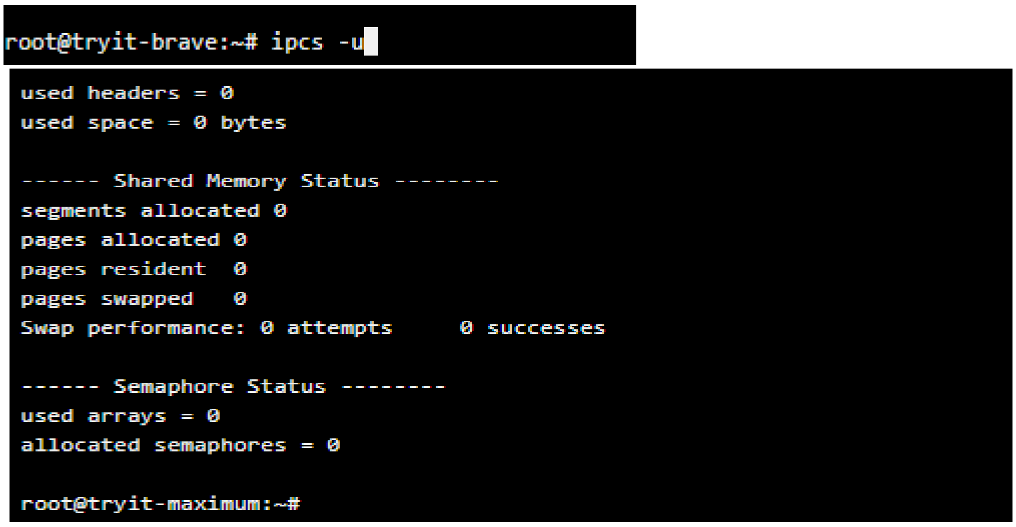

```
root@tryit-maximum:~# ipcs -m -l

------ Shared Memory Limits --------
max number of segments = 4096
max seg size (kbytes) = 18014398509465599
max total shared memory (kbytes) = 18014398509481980
min seg size (bytes) = 1

root@tryit-maximum:~#
```

**Here, "-m -l" - Option is use to print the system limits for each IPC facilities.**

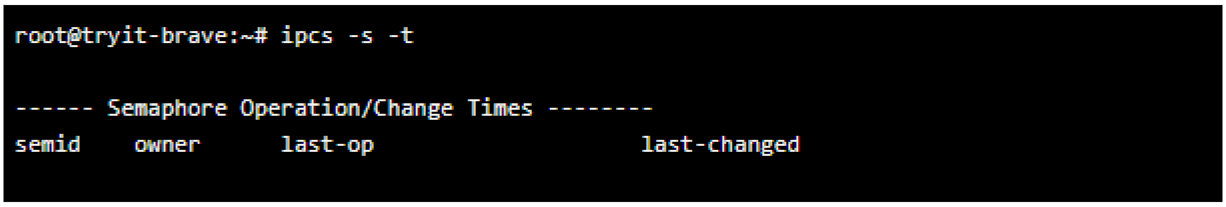

```
root@tryit-brave:~# ipcs -l

------ Shared Memory Limits --------
max number of segments = 4096
max seg size (kbytes) = 18014398509465599
max total shared memory (kbytes) = 18014398509481980
min seg size (bytes) = 1

------ Semaphore Limits --------
max number of arrays = 32000
max semaphores per array = 32000
max semaphores system wide = 1024000000
max ops per semop call = 500
semaphore max value = 32767
```

**Here, "-m -l" - Option is use to print the system limits for all 3 IPC facilities.**

**3) List status of current usage:-**


```
root@tryit-brave:~# ipcs -u
used headers = 0
used space = 0 bytes

------ Shared Memory Status --------
segments allocated 0
pages allocated 0
pages resident  0
pages swapped   0
Swap performance: 0 attempts     0 successes

------ Semaphore Status --------
used arrays = 0
allocated semaphores = 0

root@tryit-maximum:~#
```
Here, "-u" **- Option is used to display current usage for all the IPC facilities.**

**4) To get the last accessed time**



```
root@tryit-brave:~# ipcs -s -t

------ Semaphore Operation/Change Times --------
semid     owner      last-op                    last-changed
```

**Here, "-s -t" -** Option is use to print the last operation time in each IPC facilities.

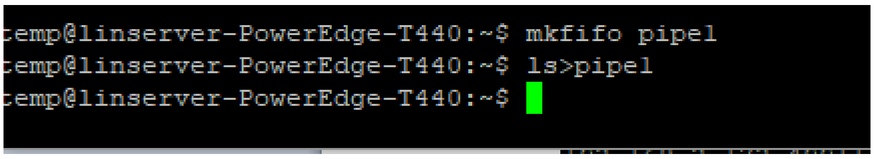
## 3.4 Two Way Communication

1) Creating a two way communication using named pipe and establishing communication between two terminals using it.

   **Syntax**:

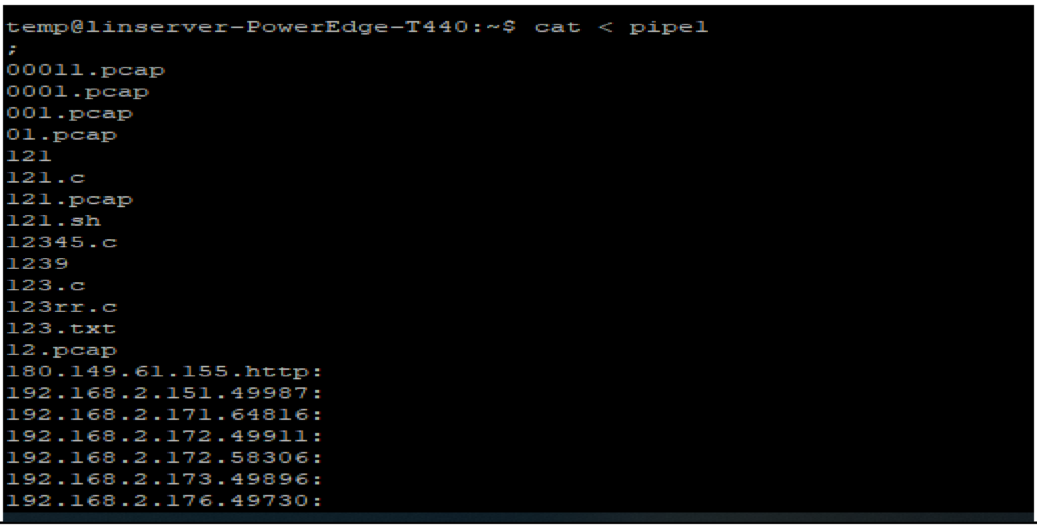
   mkfifo [pipename]

   ls>pipename



```
temp@linserver-PowerEdge-T440:~$ mkfifo pipel
temp@linserver-PowerEdge-T440:~$ ls>pipel
temp@linserver-PowerEdge-T440:~$ 
```

2) On second terminal when cat < pipename is written, it displays the output contained by that named pipe. In this way, two way communication is established between two terminals using named pipe.



```
temp@linserver-PowerEdge-T440:~$ cat < pipel
;
00011.pcap
0001.pcap
001.pcap
01.pcap
121
121.c
121.pcap
121.sh
12345.c
1239
123.c
123rr.c
123.txt
12.pcap
180.149.61.155.http:
192.168.2.151.49987:
192.168.2.171.64816:
192.168.2.172.49911:
192.168.2.172.58306:
192.168.2.173.49896:
192.168.2.176.49730:
```

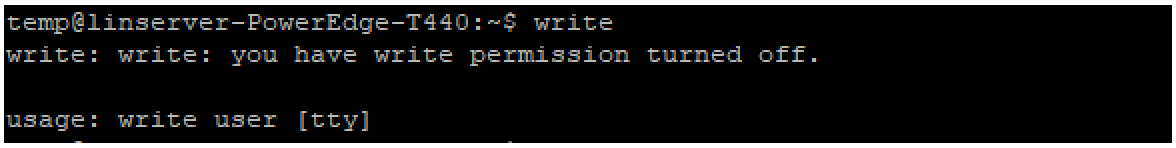
## 3.5 write COMMAND

This command is used to send message to another user. It allows you to communicate with other users, by copying lines from one user's terminal to other user's terminal.

If the user you want to write to, is logged in on more than one terminal, you can specify which terminal to write to by specifying the terminal name as the second operand to the write command.

**Syntax:**

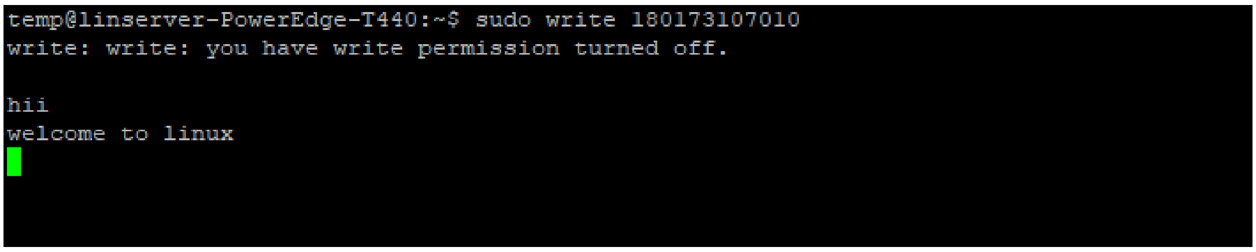
```
write user [tty]
```



```
temp@linserver-PowerEdge-T440:~$ write
write: write: you have write permission turned off.

usage: write user [tty]
```

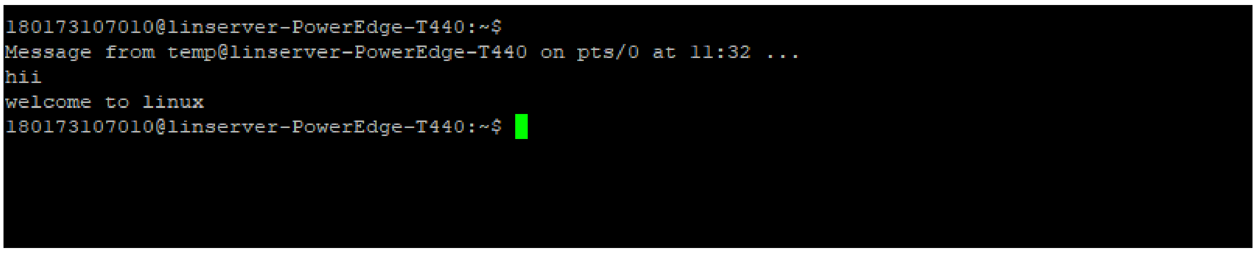
**Example:**

write metal: In this command, "metal" is the name of another user that is logged in the terminal and when this command is executed, it displays a message that gets notification on the terminal showing that it received a message from another user.



```
temp@linserver-PowerEdge-T440:~$ sudo write 180173107010
write: write: you have write permission turned off.

hii
welcome to linux
```

The response received from that user



```
180173107010@linserver-PowerEdge-T440:~$
Message from temp@linserver-PowerEdge-T440 on pts/0 at 11:32 ...
hii
welcome to linux
180173107010@linserver-PowerEdge-T440:~$
```

The basic use of this command is for sending the messages to the users available on the other terminals. All the users that are logged in will get a pop-up messages once we apply the write command and type our message.

You will also receive a message from this particular user so if any other user wants to broadcast his message he can do the same.

## 3.6 mail COMMAND

This command in linux or any unix based system is used for sending emails to the users on the system. Also, you can read and delete the received emails.

For writing an automated scripts, this command proves to be very useful. For example, an automated script written for taking periodical backup of the database i.e., sending an email from the automated script at the end of the backup for knowing the status.

**Example:**

```
$ mail -s "Hello World" someone@example.com

$ mail -s "Hello World" someone@example.com

Cc:

Hi Peter

How are you

I am fine

Good Bye
```

**Options**

-v : Verbose mode. Delivery details are displayed on the terminal.

-s : Specify the subject of the mail

-c :Send carbon copies of the mail to the list of users. This is like cc option in Microsoft outlook.

-b :Send blind copies of the mail to the list of users. This is like bcc option in outlook.

-f : Read the contents of the mailbox

-r: Specify the “from address” in send mail options.

**Examples:**

1. **Sending sample email to user**

The basic functionality of the mail command in unix or linux system is to send an email to the user.

echo "Mail body" | mail -s "Mail subject" user@user.com

- This command is used to send an email to someone@example.com. The -s option is used to specify the subject of the mail followed by the recipient email address.
- Once the above command is written, upon hitting Enter, Next you have to type in the message. When you are done with this, press 'Ctrl+D' at the beginning of a line.

Here the echo statement is used for specifying the body of the email. The -s option is used for specifying the mail subject. The mail command sends the email to the user user@user.com

2. **Specifying the body in a file.**

If we want to write a lengthy text in the mail the specifying the body with the echo statement is a tedious process. Hence, you can prepare the text file containing the body of the mail. We can redirect the contents of the text file to mail command using one of the following options.

Here the body.txt file contains the body of the email.

3. **Send mail to more than one user**

   You can send email to more than one user by specifying the users in comma separated list.

   mail -s "Mail subject" "user1@example.com,user2@example.com" < body.txt

4. **Using the cc and bcc option**

   You can copy the emails to more number of users by using the -c and -b options. An example is shown below:

   mail -s "Mail subject" -c "ccuser@gmail.com" -b "bccuser@yahoo.com" "user@example.com" < body.txt

5. **Specifying the from address**.

To send the emails with the “from address” as the logged in user. You can explicitly specify the from-address using the -r option.

```
cat body.txt | mail -s "Mail subject" "to-user@example.com" -- -r "from-
user@example.com"
```

6. **Attaching files.**

The mail command does not provide an option for attaching files. There is a workaround for attaching files using the uuencode command. Pipe the output of uuencode command for attaching files.

```
uuencode attachment-file | mail -s "Mail subject" "to-user@example.com" <
body.txt
```

**7. Reading Emails:**

viewing all the received emails

Simply type the mail and then press enter to view the received emails.

```
$Mail
```

Another way of viewing the emails is using the -f option. This is shown below:

```
$ mail -f /var/spool/mail/user
```

## 3.7 finger COMMAND

**Syntax:-**

finger [-lmsp] [*user* ...] [*user@host* ...]

This command can help the user to obtain the information about the user on its network and along with that **who**command can be used to see who are currently logged on your system. The who command list all users currently connected, along with when, how long, and where they logged in. Even though most systems for the security reason blocks the finger command, it can operate on large networks.

- Display information about the user ch. Output will appear similar to the following

**Example:**

```
finger -p ch
```

**(Output):**

- Login name: temp
- In real life: Operating System
- On since Feb 10 22:35:14 on pts/7 from domain.computeros.com
- 24 seconds Idle Time
- Unread mail since Mon Feb 11 00:21:40 2000

We can make finger display column wise by supplying –s command line option. Here's an example.

```
finger –s temp
```

**Output:**

```
Login   Name    Tty     Idle    Login   Time    Office  Office Phone
 Temp   temp    pts/7   26 sec  Feb 10 22:37    (:0)
```

Finger command will provide great help depending upon the work which you do on your linux box.

## 3.8 TELNET: REMOTELOGIN

Telnet is a command and an underlying TCP/IP protocol for accessing remote computers. It stands for telephone network

It acts as an interface to the telnet protocol i.e., allows the telnet clients to access the resources on the telnet.

Telnet utility is one of the important utility, which is used to connect with remote machine and work on that machine remotely.

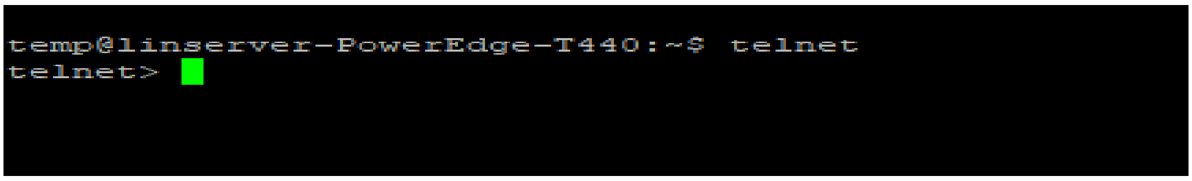
### How Does Telnet Work?

- Telnet works on a terminal.

- These computers requires keyboard because everything on the screen is displayed as text.
- Telnet provides a way to remotely log on to another device, and work on that device.
- No graphical UI to see with modern computers and operating systems.

## TELNET Command:

- Enter telnet in command mode and the prompt of telnet is given as shown below.



```
temp@linserver-PowerEdge-T440:~$ telnet
telnet>
```

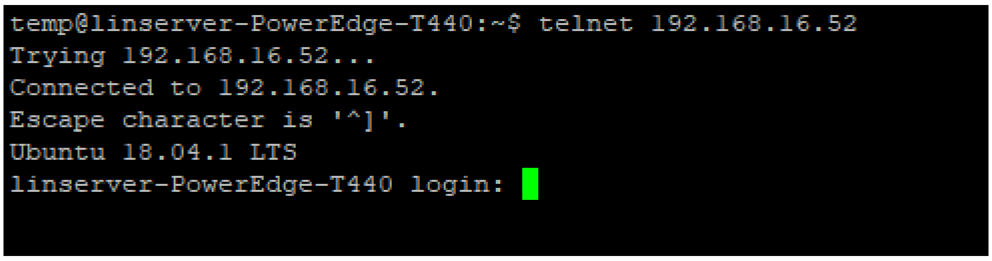
## Connecting to the telnet server:

The basic question is how the user will connect with telnet. User can connect with the Telnet by using the hostname or ip of the host.

**Syntex 1:**

$telnet Hostname/Ip Address

**Example**:



```
temp@linserver-PowerEdge-T440:~$ telnet 192.168.16.52
Trying 192.168.16.52...
Connected to 192.168.16.52.
Escape character is '^]'.
Ubuntu 18.04.1 LTS
linserver-PowerEdge-T440 login:
```

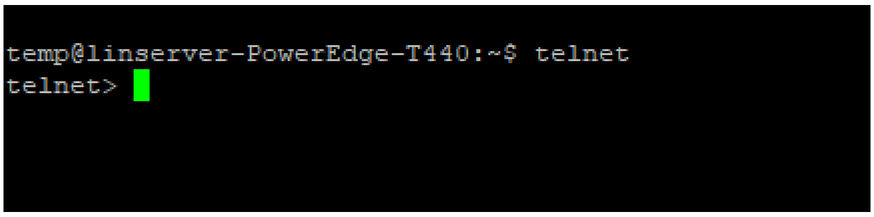
It will directly connect with telnet server. The telnet command internally executes the open command without any option.

**Syntax**2:

Without using Hostname or Ip Address

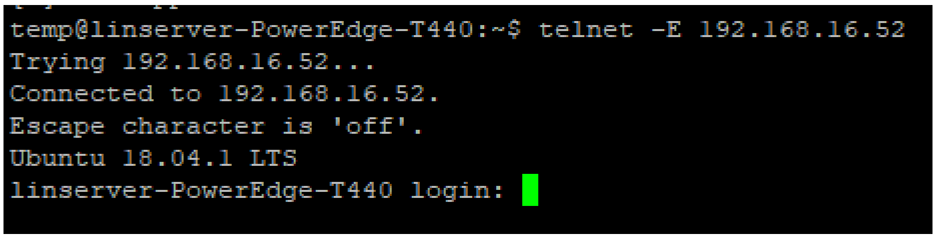
**Example:**

Connecting to telnet



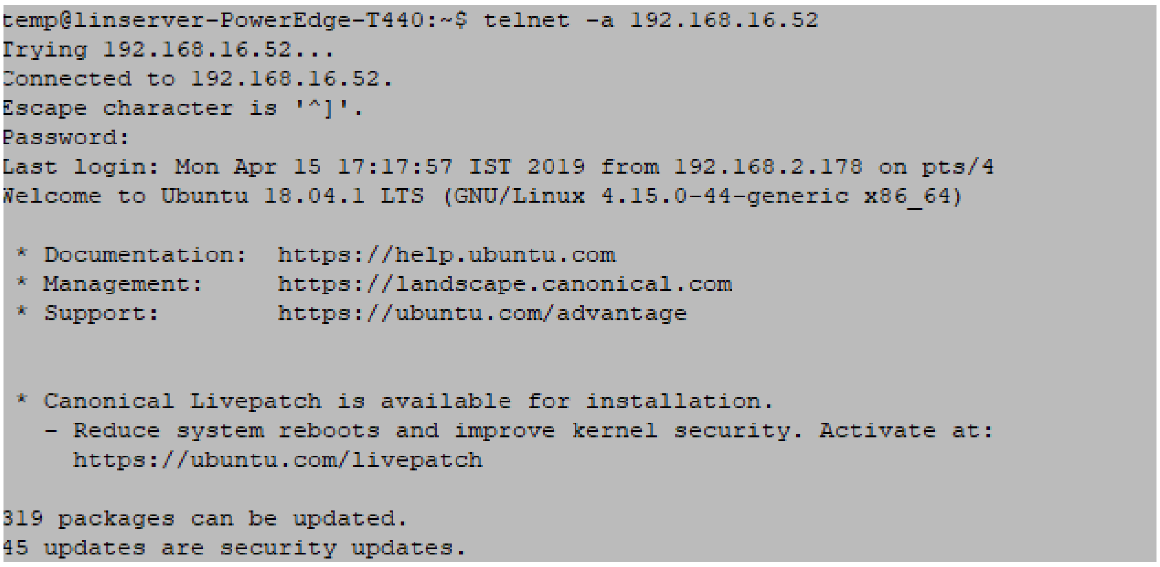
```
temp@linserver-PowerEdge-T440:~$ telnet
telnet>
```

**Disabling the escape character**



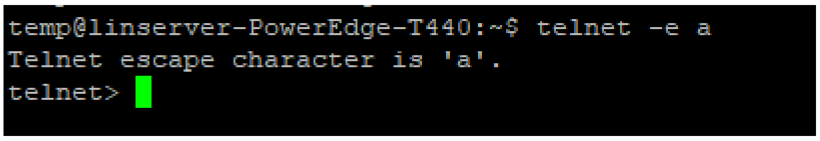
```
temp@linserver-PowerEdge-T440:~$ telnet -E 192.168.16.52
Trying 192.168.16.52...
Connected to 192.168.16.52.
Escape character is 'off'.
Ubuntu 18.04.1 LTS
linserver-PowerEdge-T440 login:
```

**Passing the user name as environment variable using -a option**



```
temp@linserver-PowerEdge-T440:~$ telnet -a 192.168.16.52
Trying 192.168.16.52...
Connected to 192.168.16.52.
Escape character is '^]'.
Password:
Last login: Mon Apr 15 17:17:57 IST 2019 from 192.168.2.178 on pts/4
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-44-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage


 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

319 packages can be updated.
45 updates are security updates.
```

**Setting the telnet escape character**



```
temp@linserver-PowerEdge-T440:~$ telnet -e a
Telnet escape character is 'a'.
telnet>
```

**1) Check telnet support commands for particular user**

**Syntax/Example:-**

$telnet help

This command will show you the different supporting commands of telnet user

## 3.9 LET US SUM UP

**Points to ponder**

- Inter-process communication is the method that allows process to communicate with each other.
- To solve the critical section problem, Semaphores have been introduced which are the integer variables that control the concurrent access by multiple process in the multitasking operating system.
- There are two types of semaphores: binary semaphores and counting semaphores.
- Binary semaphore is used for the mutual exclusion.
- Counting semaphore is used to coordinate the resource accesses.
- ipcs shows information on the inter-process communication facilities
- To connect to a remote Linux computer and work on it, telnet utility can be used.
- It provides a way to remotely log on to another device, and work on that device. This method of communication is done via Telnet.

## 3.10CHECK YOUR PROGRESS

**State true or false**

1. Finger command is used to print the information of the operating system.
2. Semaphore is a synchronization tool for accessing the shared resources.
3. Write command is used to mail other user on the system
4. Pipe operator is used for redirecting the output of the command to the other command.
5. Telnet is an application layer protocol.

## 3.11FURTHER READING

1) Here is an article about IPC in brief: https://www.w3schools.in/operating-system-tutorial/interprocess-communication-ipc/
2) Here is an article about Telnet in brief: https://www.guru99.com/communication-in-linux.html

## 3.12ASSIGNMENTS

1. Learn About Producer-Consumer Problem in IPC.
2. Try to explore more about mail command.
3. Learn Two-way communication in detail.
4. Learn about Client-server FIFO-architecture.

## 3.13 ACTIVITIES

1) Implement Producer Consumer problem using Threads.
   Reference link (https://www.thecrazyprogrammer.com/2016/09/producer-consumer-problem-c.html).

# Unit 4: Shell Programming

4

## Unit Structure

4.10. Learning Objectives

4.11. Introduction

4.12. VI Editor Command

4.13. Shell Scripts

4.14. Command Line Arguments

4.15. Shells and Sub-Shells

4.16. Shell Functions

4.17. String Handling

4.18. Array

4.19. Let us sum up

4.20. Further Reading

4.21. Assignments

4.22. Activities

## 4.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Working with VI editor
- Learning basics about shell script
- Command Line Arguments
- String Handling

## 4.2 INTRODUCTION

This unit discusses about the vi editor used in LINUX terminal. We also discuss about various content of shell scripts and some examples of shell scripts. Also we discuss some of content of string handling in Linux.

## 4.3 VI EDITOR COMMAND

- The vi editor is elaborated as visual editor.
- It is installed in every Unix system.
- It is a very powerful application.
- An improved version of vi editor is **vim**but most Linux systems have vi editor installed.

**Syntax :**

```
vi<filename>
```

➢ **vi editor has three modes:**

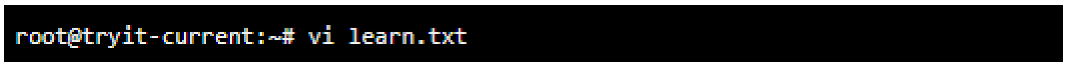
- **Command Mode:** It is command mode or default mode for various kind. It can be executed in this mode to perform various task on file.
- **Insert Mode:** In insert mode, entered text will be inserted into the file.
- **Press Esc** key will take you to the command mode from insert mode.

- **Command line Mode :**Inthis mode , first press the ESC key and then command starting with by adding **':'(colon) and command name** .

| **Command** | **Description** |
|---|---|
| i | Activating the insert mode |
| R | Opens an existing file in read only mode. |
| ZZ | ZZ command works the same way as the :wq command.(save & exit form vi) |
| :q | Command is used to quit out of vi |
| :q! | This command is used to get exit from vi without saving any of the changes. |
| :w | The command to save the contents of the editor is |
| :wq | In this command combine with the quit and return to terminal (save & exit form vi). |

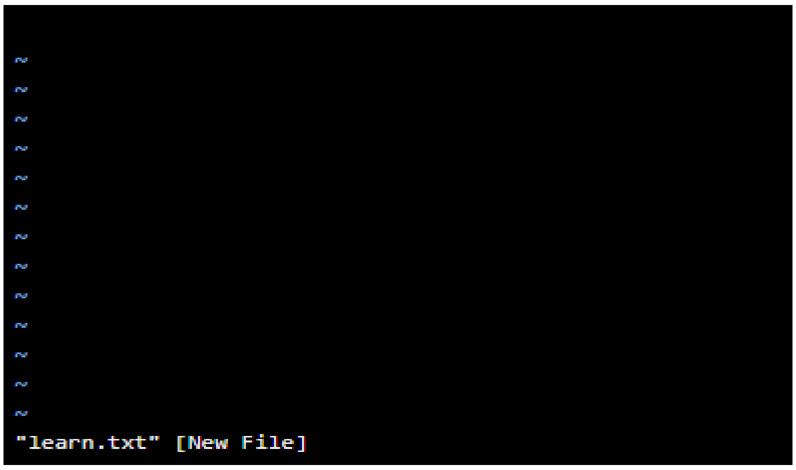
**Example :**

Open vi editor:



```
root@tryit-current:~# vi learn.txt
```

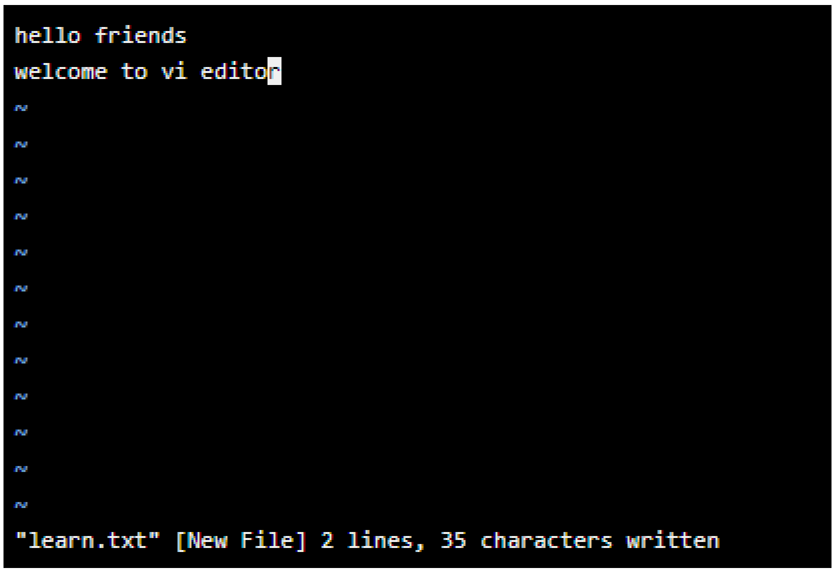
**1)Command mode**



```
~
~
~
~
~
~
~
~
~
~
~
~
~
"learn.txt" [New File]
```

In the above Snap, Blank vi editor will display. Press “**i”** to move into Insert mode.At below in vi editor filename is display.

**2)Insert mode**



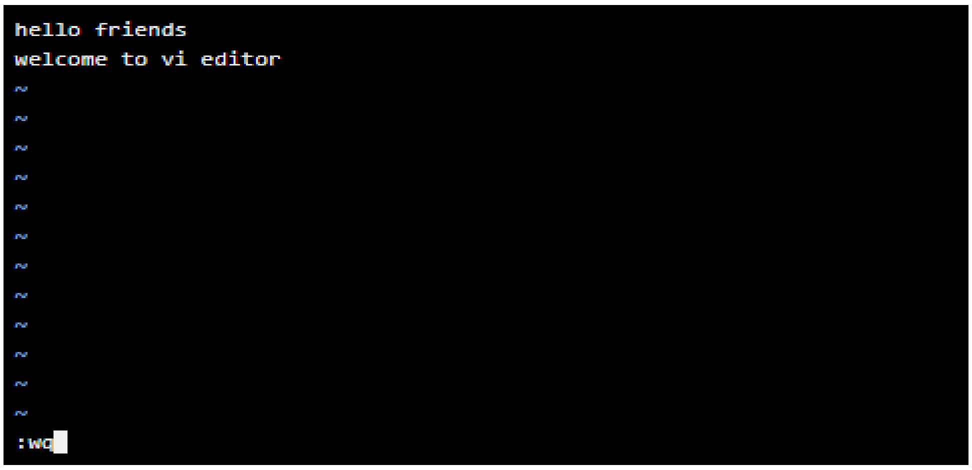
```
hello friends
welcome to vi editor
~
~
~
~
~
~
~
~
~
~
~
~
"learn.txt" [New File] 2 lines, 35 characters written
```

To move to the insert mode press i.

Start typing in vi editor and save with “ESC + :w”.after complete writing in editor.

**3) Command line mode**

Press “ESC + :wq”(save and exit from vi editor)



```
hello friends
welcome to vi editor
~
~
~
~
~
~
~
~
~
~
~
~
:wq
```

As,we type “ESC + :wq” the data inside vi editor is save and exit from vi editor and move to unix terminal.

## 4.4 SHELL SCRIPTS

**About Shell:**

The shell is a service program through which the user communicates with the OS and which is responsible for the interpretation of the input commands.

**Different UNIX shells:**

Since the shell does not directly belong to the OS, a number of different shells have been developed in the course of time:

- Bourne shell (sh). A well-known and widespread shell, named after its inventor Steven Bourne. An advanced derivate, the bash, Bourne again shell(note the pun) is most popular under Linux.
- C-Shell (csh). Developed in Berkeley, and uses a more C-like syntax. An improved version of the C-shell is the tcsh.
- Bash shell (bash). Advanced Bourne shell and standard on many systems.

**Introduction to Shell scripts**

A shell scripts also known as "shell programming".

Shell scripts are small programs consisting of UNIX commands and shell-specific program constructs (branches, loops etc), which behave like UNIX commands but are present in text form (instead of binary).

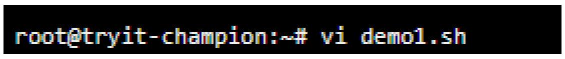
These scripts are interpreted by the shell.The syntax of shell scripts differs (considerably) from shell to shell.

Let's, take example:

A shell program to print hello world, compile file and run the file content.

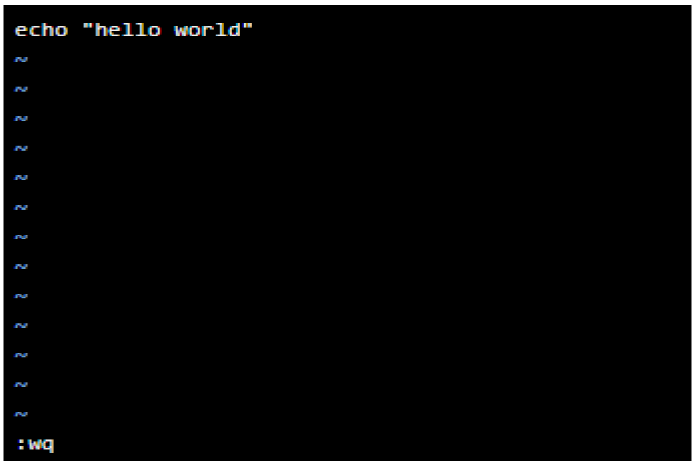
We will recall the above steps:

**Step 1**: Create the file with (**.sh**) extension.



```
root@tryit-champion:~# vi demo1.sh
```

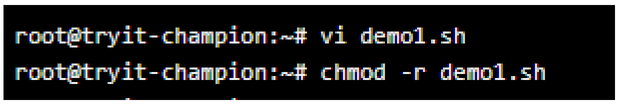
**Step 2**: Type the “hello world” in echo command.as shown below and save and exit with **“ESC + :wq”.**



```
echo "hello world"
~
~
~
~
~
~
~
~
~
~
~
~
~
:wq
```

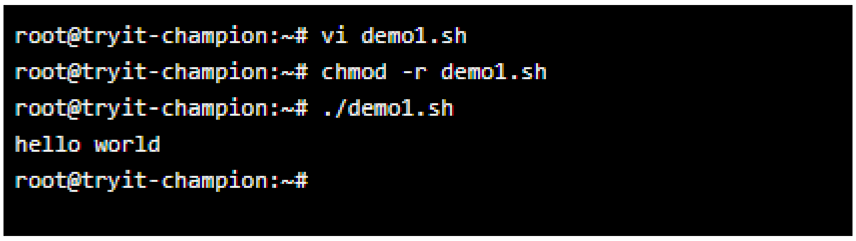
**Step 3**:For compile the program:

Write :**“chmod  -r <filename>”**



```
root@tryit-champion:~# vi demo1.sh
root@tryit-champion:~# chmod -r demo1.sh
```

**Step 4**: After compile to run the program **“./<filename>”**

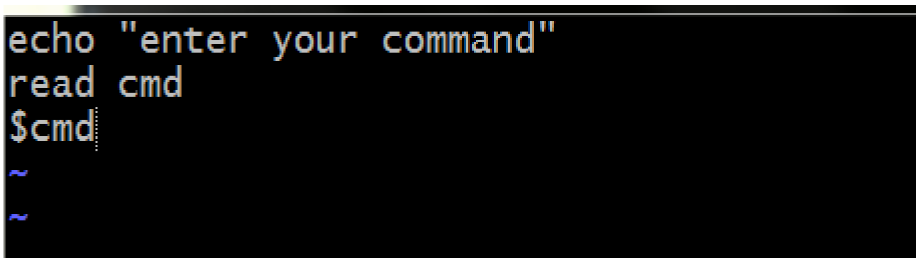


```
root@tryit-champion:~# vi demo1.sh
root@tryit-champion:~# chmod -r demo1.sh
root@tryit-champion:~# ./demo1.sh
hello world
root@tryit-champion:~#
```

**Shell scripts Examples**

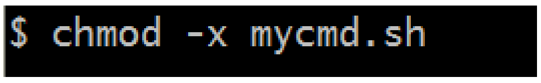
**Aim**: Create a shell script which scans the command name and execute command output.

**Program:**

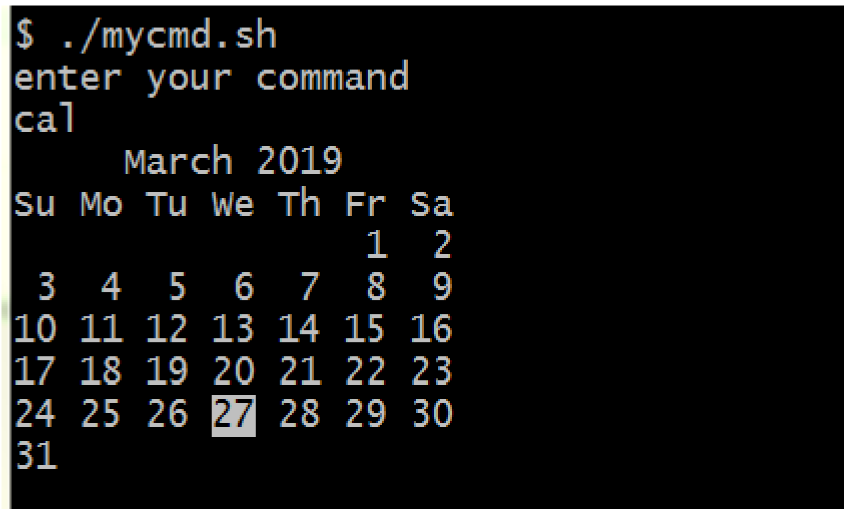


```
echo "enter your command"
read cmd
$cmd
~
~
```

**Output:**



```
$ chmod -x mycmd.sh
```



```
$ ./mycmd.sh
enter your command
cal
     March 2019
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
```

## 4.5COMMAND LINE ARGUMENTS

- List the arguments on the command line when running a shell script known as Command line arguments.
- Command line arguments (also known as positional parameters) are the arguments specified at the command prompt with a command or script to be executed.
- The command-line arguments $1, $2, $3, ...$9 are positional parameters and

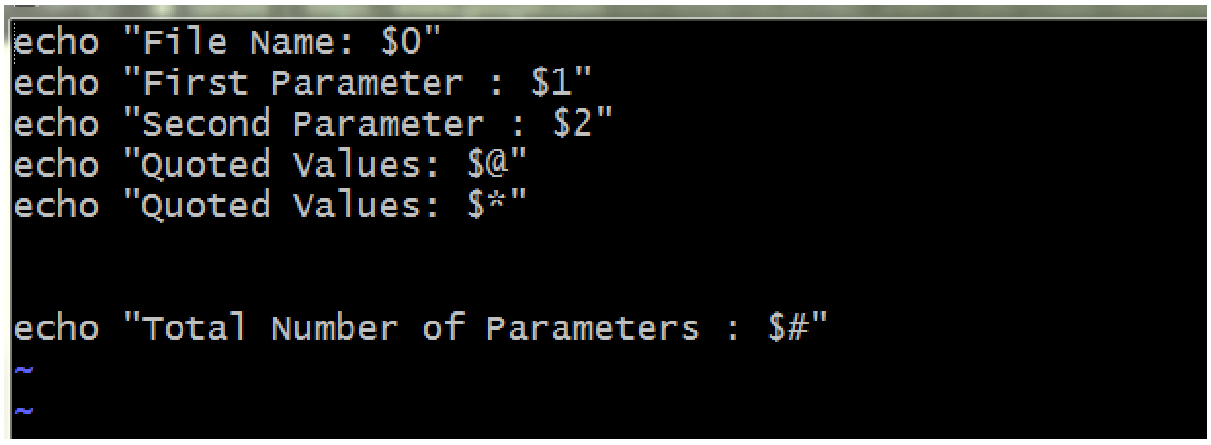
| **Variables** | **Descriptions** |
|---|---|
| $1, $2, $3, ...$9 | Display positional parameters |
| $0 | Used to display the filename of the current script. |
| $n | These variables correspond to the arguments with which a script was invoked.<br>Here n is a positive decimal number corresponding to the position of an argument (the first argument is $1, the second argument is $2, and so on). |

| | |
|---|---|
| $@ | $@ is equivalent to $1 $2. |
| $# | The number of arguments supplied to a script. |
| $* | All the arguments are double quoted. If a script receives two arguments, $* is equivalent to $1 $2. |

**Example:**

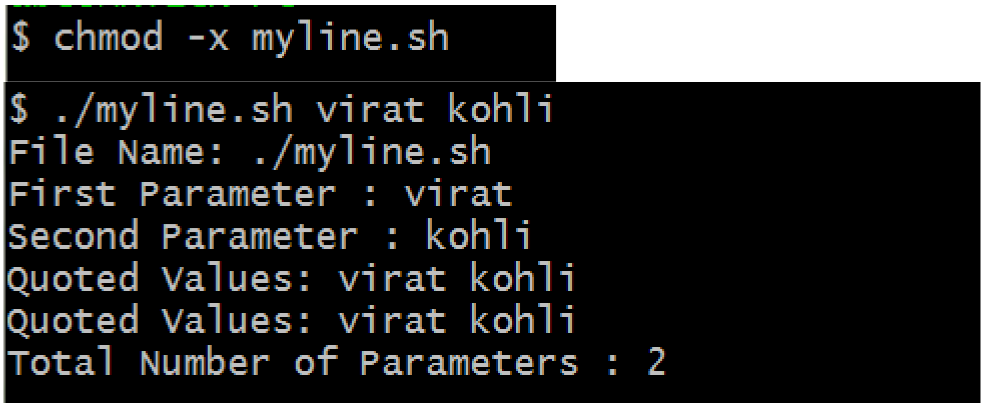
**Aim:** Write a shell script which display the use of command line arguments.

**Program:**



```
echo "File Name: $0"
echo "First Parameter : $1"
echo "Second Parameter : $2"
echo "Quoted Values: $@"
echo "Quoted Values: $*"


echo "Total Number of Parameters : $#"
~
~
```

**Output:**



```
$ chmod -x myline.sh
```

```
$ ./myline.sh virat kohli
File Name: ./myline.sh
First Parameter : virat
Second Parameter : kohli
Quoted Values: virat kohli
Quoted Values: virat kohli
Total Number of Parameters : 2
```

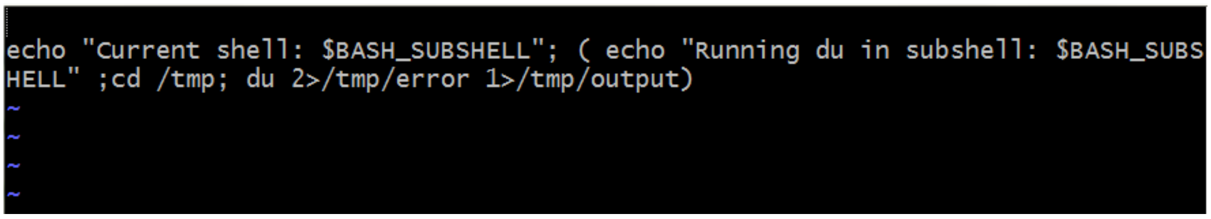
## 4.6SHELLS AND SUB-SHELLS

- As we run a shell script, it creates a new process called subshell and your script will get executed using a subshell.
- A Subshell can be used to do parallel processing.
- If we start another shell on top of your current shell, it can be referred to as a subshell.
- Type the following command to see subshell value:

**Example:**

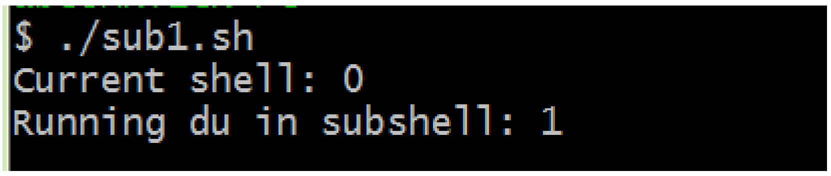
**Aim:** Write a shell script Which display the use of subshell.

**Program:**



```
echo "Current shell: $BASH_SUBSHELL"; ( echo "Running du in subshell: $BASH_SUBS
HELL" ;cd /tmp; du 2>/tmp/error 1>/tmp/output)
~
~
~
~
```

**Output:**



```
$ ./sub1.sh
Current shell: 0
Running du in subshell: 1
```

## 4.7SHELL FUNCTIONS

- Shell Functions working is similar to functions working in basic languages.
- Shell Function can be defined as a block of commands.
- This, block of commands are invoked at different stages of execution.
- Main advantages of shell function is that functions code can be reuse.

**Defining function:-**

function_name()

{

--------//code

//statements

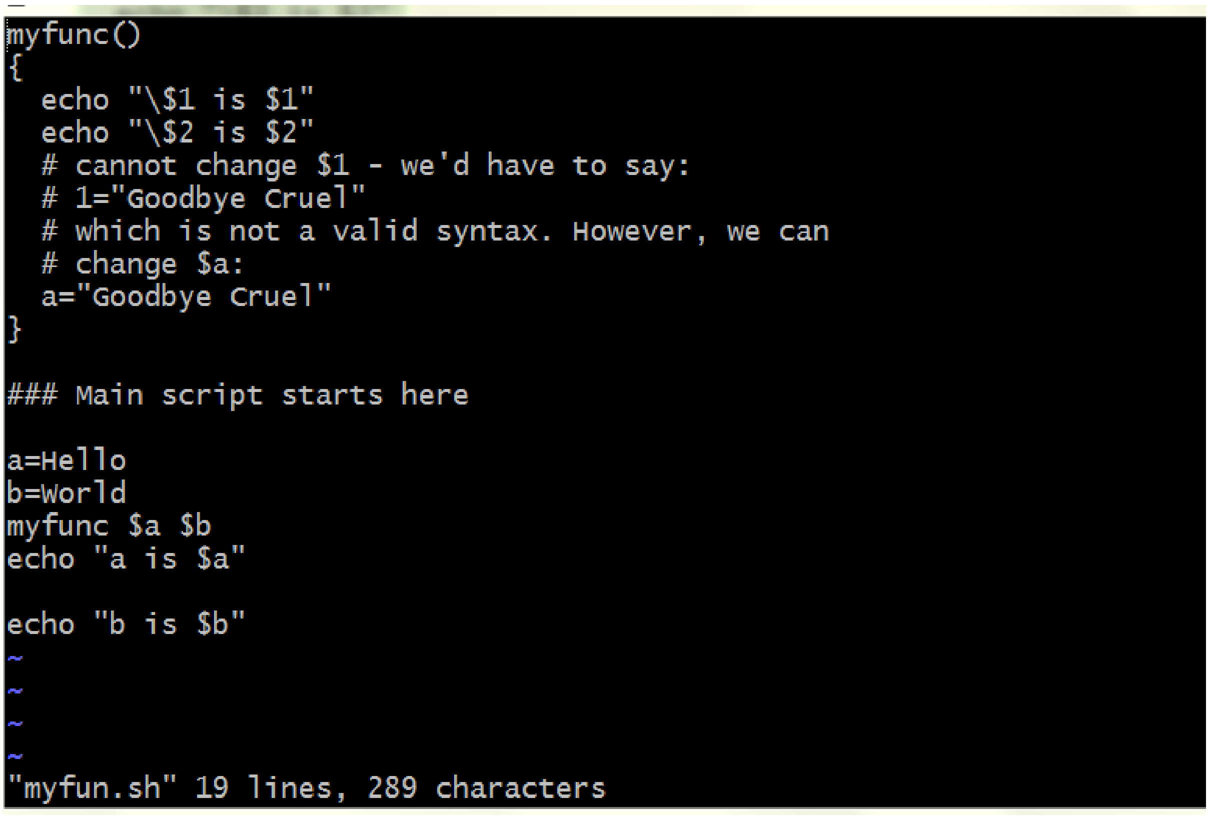
--------//code

}

**Invoking function:-**

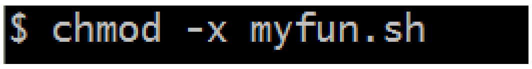
$ function_name

**Program:**

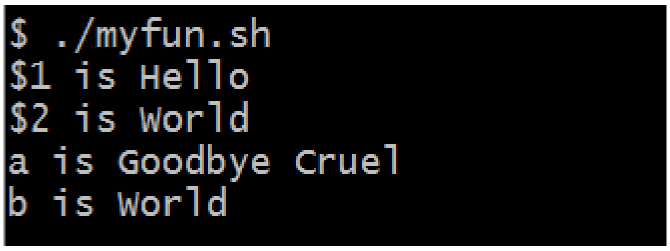


```
myfunc()
{
  echo "\$1 is $1"
  echo "\$2 is $2"
  # cannot change $1 - we'd have to say:
  # 1="Goodbye Cruel"
  # which is not a valid syntax. However, we can
  # change $a:
  a="Goodbye Cruel"
}

### Main script starts here

a=Hello
b=World
myfunc $a $b
echo "a is $a"

echo "b is $b"
~
~
~
~
"myfun.sh" 19 lines, 289 characters
```

**Output:**



```
$ chmod -x myfun.sh
```



```
$ ./myfun.sh
$1 is Hello
$2 is World
a is Goodbye Cruel
b is World
```

## 4.8STRING HANDLING

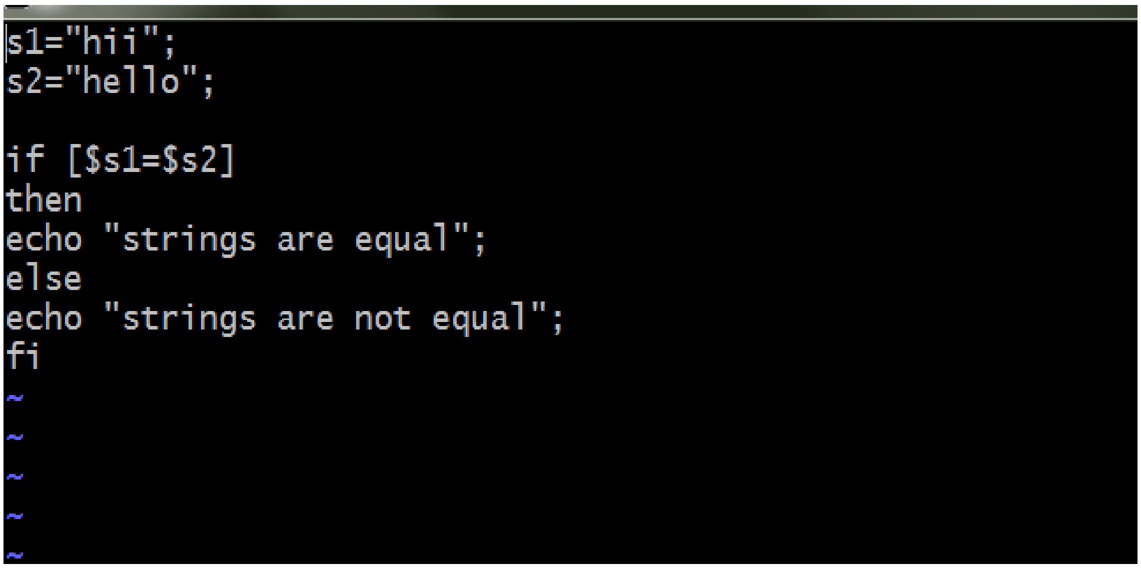
There are many operators in Shell Script some of them are discussed based on string.

1. **Equal operator (=):** This operator is used to check whether two strings are equal.

   **Syntax:**

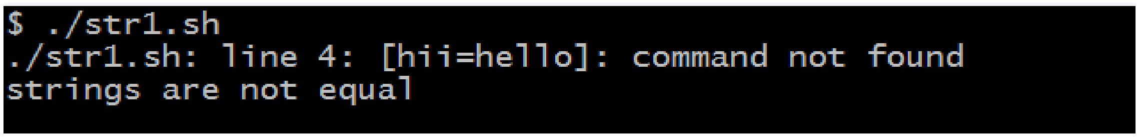
   **Operand1 = Operand2**

   **Program:**



```
s1="hii";
s2="hello";

if [$s1=$s2]
then
echo "strings are equal";
else
echo "strings are not equal";
fi
~
~
~
~
~
```

   **Output:**



```
$ ./str1.sh
./str1.sh: line 4: [hii=hello]: command not found
strings are not equal
```

**2.Not Equal operator (!=):**

   **Syntax:**

   **Operand1 != Operand2**

**3.  Less then (\<):**

   **Syntax:**

   **Operand1 \< Operand2**

**4.Greater then (\>):**

   **Syntax:**

   **Operand1 \> Operand2**

## 4.9 ARRAY

- Array is collection of similar data types.
- By default in shell script everything is treated as a string.
- An array is zero-based ie indexing start with 0.

We can declare array in a shell script in different ways.

1. Indirect Declaration
2. Explicit Declaration

**1. Indirect Declaration**

In this declaration, We assigned a value in a particular index of Array Variable

**2. Explicit Declaration**

In this declaration, First We declare array then assigned the values.

➢ **Assignment of values in array**

**Syntax :**

**Arrayname = (val1,val2,...valn)**

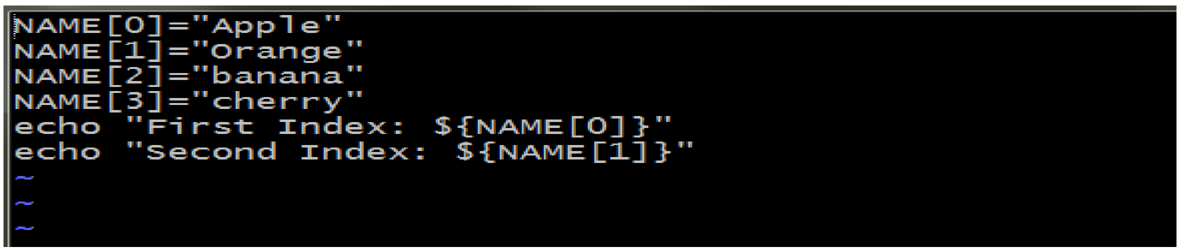
**Or**

**Arrayname = ([1]=11 [2]=22 [3]=33)**

**Example :**

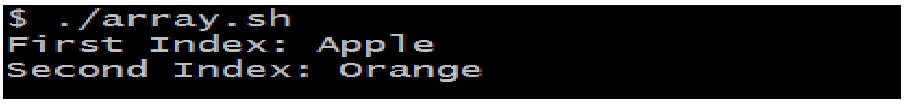
**Aim:** Write a shell script to make use of Array.

**Program:**



```
NAME[0]="Apple"
NAME[1]="Orange"
NAME[2]="banana"
NAME[3]="cherry"
echo "First Index: ${NAME[0]}"
echo "Second Index: ${NAME[1]}"
~
~
~
```

**Output:**



```
$ ./array.sh
First Index: Apple
Second Index: Orange
```

## 4.10 LET US SUM UP

**Points to ponder**

- The vi editor is elaborated as visual editor and is installed in every Unix System.
- The shell is a service program through which the user communicates with the OS and which is responsible for the interpretation of the input commands.
- Command line arguments (also known as positional parameters) are the arguments specified at the command prompt with a command or script to be executed.
- A Subshell can be used to do parallel processing.
- Shell Functions working is similar to functions working in basic languages.

## 4.11 FURTHER READING

2) Here is an article about the concept of shells in computing: http://en.wikipedia.org/wiki/Shell_(computing)
2) Shell scripting Tutorial:
   https://www.shellscript.sh/
   https://www.tutorialspoint.com/unix/shell_scripting.htm

## 4.12 ASSIGNMENTS

1. Write a shell script to calculate simple interest.
2. Write a shell script to calculate salary from given basic.

   Salary = basic + dp + da +hra +ma –pf
   basic – to be taken as input
   dp - 50 % of basic
   da - 35 % of (basic + dp)
   hra – 8 % of (basic + dp)
   ma - 3 % of (basic + dp)
   pf - 10% of (basic + dp)

3. Write a shell script to calculate the average of a set of N number.
4. Write a UNIX shell script to find the sum of number to given number. e.g. if entered number is 5 then 1+2+3+4+5
5. Write a shell script to perform like calculator. It should ask for the number and operand from the user

## 4.13 ACTIVITIES

1. Write a shell script to calculate the area of rectangle. It should take the value from the command line.
2. Write a shell script to take two numbers from command line and show result of dividing small number with bigger number. Also note that it should not accept zero or negative number. If user enter zero or negative number then it should prompt to input correct number after displaying proper message.
3. Write a Unix Shell Script which prints the following:
   a. Current home directory.
   b. Current user name.
   c. The message "Today is :" with current date in MM/dd/yy format
   d. The message "No of users logged in :" with total no of current logged in users
   e. The message "Terminal :" With you own terminal number
4. Write a shell script to create a command line calculator.

   e.g. input : mycal 5 + 5 Result : 10 , input : mycal 5 / 5 result : 1
5. Write a shell script that that takes as command line input a number N and a Word. it then prints the word n times., one word per line

BAOU
Education
for All

## યુનિવર્સિટી ગીત

સ્વાધ્યાયઃ પરમં તપઃ
સ્વાધ્યાયઃ પરમં તપઃ
સ્વાધ્યાયઃ પરમં તપઃ

શિક્ષણ, સંસ્કૃતિ, સદ્‌ભાવ, દિવ્યબોધનું ધામ
ડૉ. બાબાસાહેબ આંબેડકર ઓપન યુનિવર્સિટી નામ;
સૌને સૌની પાંખ મળે, ને સૌને સૌનું આભ,
દશે દિશામાં સ્મિત વહે હો દશે દિશે શુભ-લાભ.

અભણ રહી અજ્ઞાનના શાને, અંધકારને પીવો ?
કહે બુદ્ધ આંબેડકર કહે, તું થા તારો દીવો;
શારદીય અજવાળા પહોંચ્યાં ગુર્જર ગામે ગામ
ધ્રુવ તારકની જેમ ઝળહળે એકલવ્યની શાન.

સરસ્વતીના મયૂર તમારે ફળિયે આવી ગહેકે
અંધકારને હડસેલીને ઉજાસના ફૂલ મહેંકે;
બંધન નહીં કો સ્થાન સમયના જવું ન ઘરથી દૂર
ઘર આવી મા હરે શારદા દૈન્ય તિમિરના પૂર.

સંસ્કારોની સુગંધ મહેંકે, મન મંદિરને ધામે
સુખની ટપાલ પહોંચે સૌને પોતાને સરનામે;
સમાજ કેરે દરિયે હાંકી શિક્ષણ કેરું વહાણ,
આવો કરીયે આપણ સૌ
ભવ્ય રાષ્ટ્ર નિર્માણ...
દિવ્ય રાષ્ટ્ર નિર્માણ...
ભવ્ય રાષ્ટ્ર નિર્માણ

**DR. BABASAHEB AMBEDKAR OPEN UNIVERSITY**
(Established by Government of Gujarat)
'Jyotirmay' Parisar,
Sarkhej-Gandhinagar Highway, Chharodi, Ahmedabad-382 481
**Website : www.baou.edu.in | Email : office.scs@baou.edu.in**

978- 81- 942146- 3-2