Chapter- 1 INTRODUCTION TO OPERATING SYSTEMS

Need for Operating System

An OS is an intermediary between the user of the computer & the computer hardware. It provides a basis for application program & acts as an intermediary between user of computer & computer hardware. The purpose of an OS is to provide an environment in which the user can execute the program in a convenient & efficient manner.

OS is an important part of almost every computer systems. A computer system can be roughly divided into four components

- The Hardware
- The OS
- The application Program
- The user

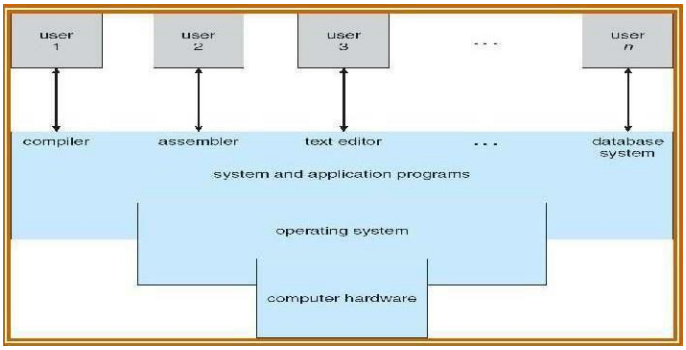
The Hardware consists of memory, CPU, ALU, I/O devices, peripherals devices & storage devices.

The application program mainly consisted of word processors, spread sheets, compilers & web browsers defines the ways in which the resources are used to solve the problems of the users.

The OS controls & co-ordinates the use of hardware among various application program for various users.

COMPUTER SYSTEM ORGANIZATION

The following figure shows the conceptual view of a computer system



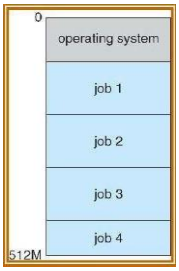
OPERATING SYSTEM ARCHITECTURE

<u>Multi programmed System</u>:

a. If there are two or more programs in the memory at the same time sharing the processor, this is referred as multi programmed OS.
b. It increases the CPU utilization by organizing the jobs so that the CPU will always have one job to execute.
c. Jobs entering the systems are kept in memory.
d. OS picks the job from memory & it executes it.
e. Having several jobs in the memory at the same time requires some form of memory management.
f. Multi programmed systems monitors the state of all active program and system resources and ensures that CPU is never idle until there are no jobs.
g. While executing a particular job, if the job has to wait for any task like I/O operation to be complete then the CPU will switch to some other jobs and starts executing it and when the first job finishes waiting the CPU will switch back to that.

h. This will keep the CPU & I/O utilization busy. The following figure shows the memory

layout of multi programmed OS



Time sharing Systems:

a. Time sharing system or multi-tasking is logical extension of multi programming systems. The CPU executes multiple jobs by switching between them but the switching occurs so frequently that user can interact with each program while it is running.

b. An interactive & hands on system provides direct communication between the user and the system. The user can give the instruction to the OS or program directly through key board or mouse and waits for immediate results.

c. A time shared system allows multiple users to use the computer simultaneously. Since each action or commands are short in time shared systems only a small CPU time will be available for each of the user.

d. A time shared systems uses CPU scheduling and multi programming to provide each user a small portion of time shared computers. When a process executes it will be executing for a short time before it finishes or need to perform I/O. I/O is interactive i.e. O/P is to a display for the user and the I/O is from a keyboard, mouse etc.

e. Since it has to maintain several jobs at a time, system should have memory management & protection.

f. Time sharing systems are complex than the multi programmed systems. Since several jobs are kept in memory they need memory management and protection. To obtain less response time jobs are swapped in and out of main memory to disk. So disk will serve as backing store for main memory. This can be achieved by using a technique called virtual memory that allows for the execution of job i.e. not completes in memory.

Time sharing system should also provide a file system & file system resides on collection of disks so this need disk management. It supports concurrent execution, job synchronization & communication.

OS Types

DISTRIBUTED SYSTEMS

- A distributed system is one in which H/w or S/w components located at the networked computers to communicate & coordinate their actions only by passing messages.
- A distributed systems looks to its user like an ordinary OS but runs on multiple, Independent CPU**'s.**
- Distributed systems depends on networking for their functionality which allows for communication so that distributed systems are able to share computational tasks and provides rich set of features to users.
- N/w may vary by the protocols used, distance between nodes & transport media. Protocols- TCP/IP, ATM etc. Network-> LAN, MAN, WAN etc. Transport Media-> copper wires, optical fibers & wireless transmissions

Advantages of Distributed Systems:

1. Resource sharing. 2. Higher reliability. 3. Better price performance ratio. 4. Shorter response time. 5. Higher throughput. 6. Incremental growth.

Real time system

Real time system is one that must react to I/p & responds to them quickly. A real time system should not be late in response to one event.

A real time should have well defined time constraints. Real time systems are of two types

Hard Real Time Systems: guarantees that the critical tasks to be completed on time.

Soft Real Time Systems: less restrictive one where a critical real time task gets priority over other tasks & retains the property until it completes.

Network operating systems

Network operating systems are installed on a server providing users with the capability to manage data, user groups and applications. This operating system enables users to access and share files and devices such as printers, security software and other applications, mostly in a local area network.

Mobile operating systems

Mobile operating systems run exclusively on small devices such as smartphones, tablets and wearables. The system combines the features of a personal computer with additional features useful for a handheld device. Mobile operating systems start when a device is powered on to provide access to installed applications. Mobile operating systems also manage wireless network connectivity.

Functions of O.S.

1. Process Management.
2. Main M/y Management.
3. File Management.
4. Secondary Storage Management.
5. I/O System management.
6. Networking.
7. Protection System.
8. Command Interpreter System.

The OS must support the following tasks

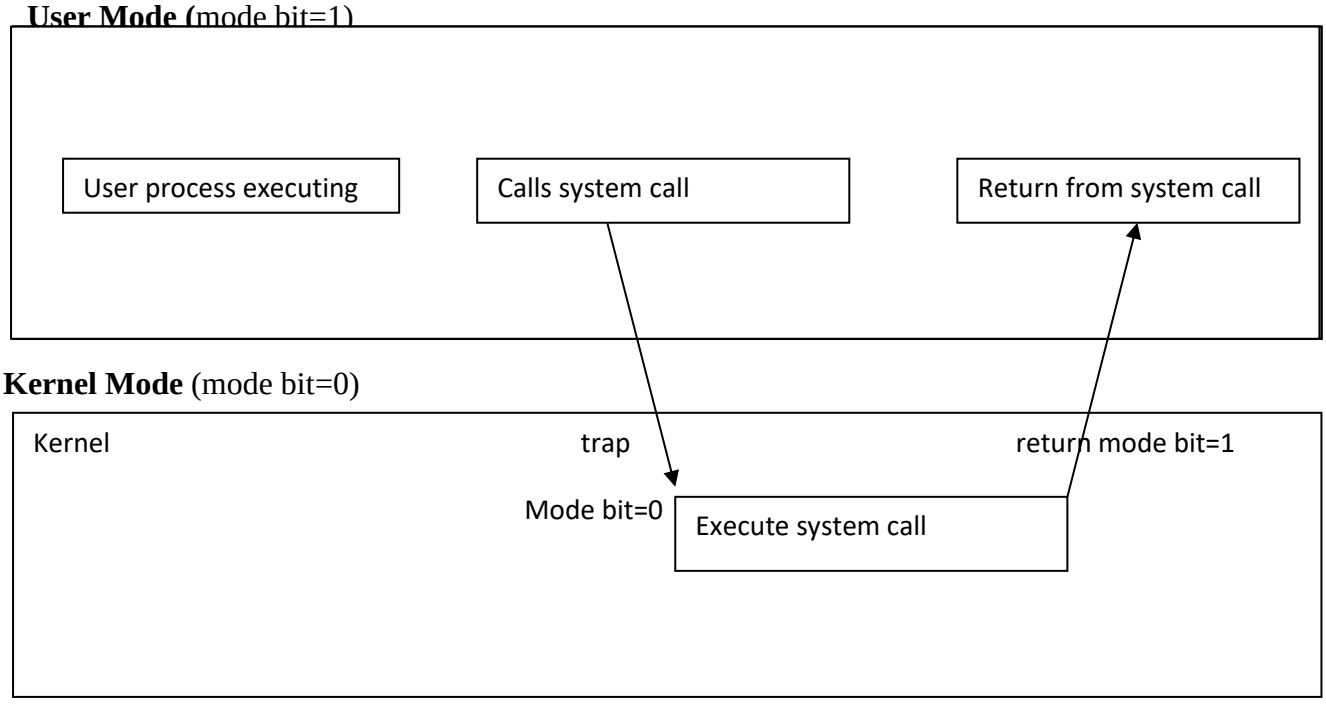
a. Provide the facility to create, modification of programs & data files using on editors.

b. Access to compilers for translating the user program from high level language to machine language.

c. Provide a loader program to move the compiled program code to computers memory for execution.

d. Provides routines that handle the details of I/O programming.

Dual mode operation



o.s execution is done in two modes.

1. Kernel Mode (Supervisor mode)
2. User Mode

A mode bit is added to the hardware of the computer to indicate current mode:

In kernel mode the mode bit is set to zero(0) and in user mode the mode bit is set to one(1).

All user applications are executed in user mode. All system related processes are executed in kernel

mode.
When system is booted, the hardware starts in kernel mode. The operating system is loaded and then user application's execution is started in user mode. Whenever there is trap or interrupt, the hardware switches from user mode to kernel mode.
Dual mode of operation provides us with the means for protecting the operating system from errant users and errant users from operating system.

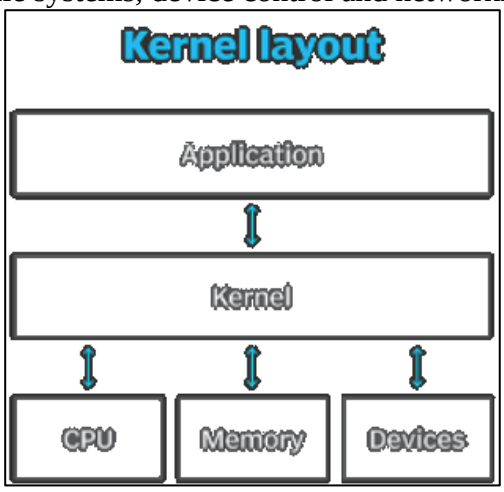
Timer

Operating system maintains a timer for user processes to avoid from process executing in infinite loop time. i.e. we can prevent a user program from running too long.
A timer can be set to interrupt after a specified period. The period may be fixed or variable. Variable timer is implemented by a fixed rate of cloak and a counter.

- The operating system sets the counter. Every time clock ticks, the counter is decremented. When the counter reaches 0, an interrupt occurs.

Kernel:

The kernel is the essential center of a computer operating system (OS). It provides basic services for all other parts of the OS. It is the main layer between the OS and hardware, and it helps with process and memory management, file systems, device control and networking.



Microkernel

In a microkernel, the user services and kernel services are implemented in different address spaces. The user services are kept in **user address space**, and kernel services are kept under **kernel address space**, thus also reduces the size of kernel and size of an operating system as well.

Responsibilities of microkernel:

- Inter process-Communication
- Memory Management
- CPU-Scheduling

Advantages of Microkernel

- The architecture of kernel is small and isolated hence it can function better.
- Expansion of the system is easier, it is simply added to the system application without disturbing the kernel.

User Interface

A User Interface (UI) performs interaction between a user and the computer.
Command line interface: It enables the users to interact with the operating system by issuing some specific commands. The user needs to type a command at the command line. When the user enters the key, the command line interpreter received a command. The software program that is responsible for receiving and processing the commands issued by the user. After processing the command are called

command line interpreter, the command line interpreter displays the command prompt again along with the output of the previous command issued by the user. The disadvantages of the CLI is that the user needs to remember a lot to interact with the operating system.

Graphical user interface: It enables the users to interact with the operating system by means of point-and-click operations.

GUI contains several icons representing pictorial representation of the variables such as a file, directory, and device. The graphical icon provided in the UI can be manipulated by the users using a suitable pointing device such as a mouse, trackball, touch screen and light pen. The other input devices like keyboard can also be used to manipulate these graphical icons. GUIs are considered to be very user-friendly interface because each object is represented with a corresponding icon. Unlike the other UIs the users need not provide text command for executing tasks.

Types of OS installation

Guided Install: Install an operating system and device drivers in an unattended mode.

Manual Install: Install an operating system and device drivers manually.

Personal needs from OS

- Provides a good interface to a single user.
- Used for word processing, spreadsheets and Internet access.
- Made only for personal.

Laptops, computer systems, tablets etc. are personal computers and the operating system such as windows 7, windows 10, android, etc. are personal computer operating system.

Personal computer operating system are used for personal purposes, for example, chatting ,using some social media sites, reading articles from internet, making some projects, designing website, programming something, watching videos and movies, listening songs and many more.

Corporate needs from OS

- Easy to use
- User friendly
- Stable.
- Does not take a lot of maintenance.

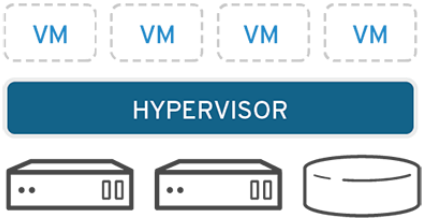
Chapter 2 Virtualization technology

Virtualization technology

It creates useful IT services using resources that are traditionally bound to hardware. It allows user to use a physical machine's full capacity by distributing its capabilities among many users or environments.

Working

Software called **hypervisors** separate the physical resources from the virtual environments. Hypervisors can sit on top of an operating system (like on a laptop) or be installed directly onto hardware (like a server), which is how most enterprises virtualize. Hypervisors take physical resources and divide them up so that virtual environments can use them.



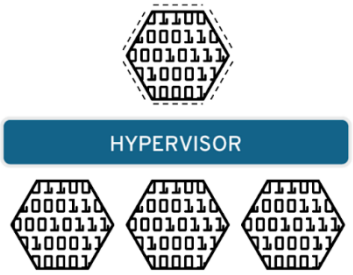
Resources are partitioned as needed from the physical environment to the many virtual environments. Users interact with and run computations within the virtual environment (typically called a guest machine or virtual machine). The virtual machine functions as a single data file. And like any digital file, it can be moved from one computer to another, opened in either one, and be expected to work the same.

When the virtual environment is running and a user or program issues an instruction that requires additional resources from the physical environment, the hypervisor relays the request to the physical system and caches the changes.

Types of virtualization

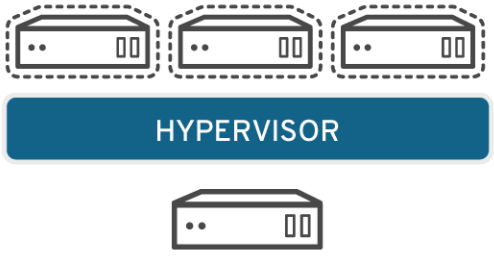
Data virtualization

Data virtualization tools sit in front of multiple data sources and allows them to be treated as single source, delivering the needed data—in the required form—at the right time to any application or user.



Server virtualization

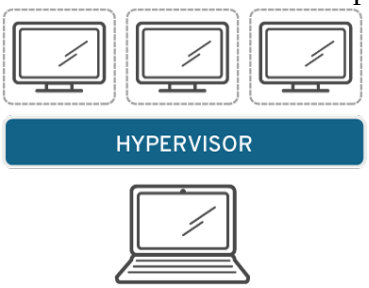
Servers are computers designed to process a high volume of specific tasks really well so other computers—like laptops and desktops—can do a variety of other tasks. Virtualizing a server lets it to do more of those specific functions and involves partitioning it so that the components can be used to serve multiple functions.



Operating system virtualization

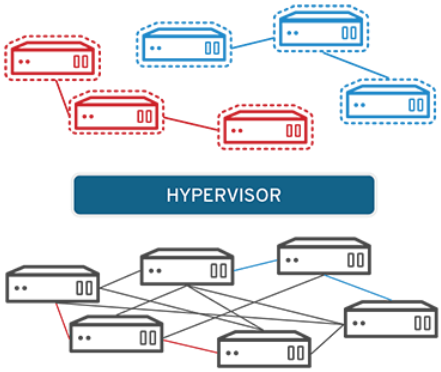
Operating system virtualization happens at the kernel—the central task managers of operating systems. It's a useful way to run Linux and Windows environments side-by-side. Enterprises can also push virtual operating systems to computers, which:

- Reduces bulk hardware costs, since the computers don't require such high out-of-the-box capabilities.
- Increases security, since all virtual instances can be monitored and isolated.
- Limits time spent on IT services like software updates.



Network functions virtualization

Network functions virtualization (NFV) separates a network's key functions (like directory services, file sharing, and IP configuration) so they can be distributed among environments. Once software functions are independent of the physical machines they once lived on, specific functions can be packaged together into a new network and assigned to an environment. Virtualizing networks reduces the number of physical components—like switches, routers, servers, cables, and hubs—that are needed to create multiple, independent networks, and it's particularly popular in the telecommunications industry.



Challenges of Virtualization

1. Determining Individual Needs
2. Licensing Restrictions
3. Resource Estimations
4. VM Management
5. Virtual Backups

Potentials of Virtualization

1. Slash your IT expenses
2. Reduce downtime and enhance resiliency in disaster recovery situations
3. Increase efficiency and productivity
4. Control independence and DevOps
5. Move to be more green-friendly (organizational and environmental)

Linux containers and virtual machines (VMs)

They are packaged computing environments that combine various IT components and isolate them from the rest of the system. Their main differences are in terms of scale and portability.

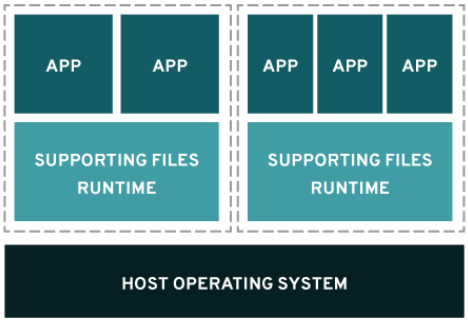
Containers are typically measured by the megabyte. They don't package anything bigger than an app and all the files necessary to run, and are often used to package single functions that perform specific tasks (known as a micro service). The lightweight nature of containers—and their shared operating system (OS)—makes them very easy to move across multiple environments

The small, lightweight nature of containers allows them to be moved easily across bare metal systems as well as public, private, hybrid, and multicolor environments. They're also the ideal environment to deploy today's cloud-native apps, which are collections of micro services designed to provide a consistent development and automated management experience across public, private, hybrid, and multicolor environments. Cloud-native apps help speed up how new apps are built, how existing ones are optimized, how they're all connected. The caveat is that containers have to be compatible with the underlying OS. Containers are best used to:

- Build cloud-native apps
- Package micro services
- Instill DevOps or CI/CD practices
- Move scalable IT projects across a diverse IT footprint that shares the same OS

Working

Containers hold a micro service or app and everything it needs to run. Everything within a container is preserved on something called an image—a code-based file that includes all libraries and dependencies. These files can be thought of as a Linux distribution installation because the image comes with RPM packages, and configuration files. Because containers are so small, there are usually hundreds of them loosely coupled together—which is why container orchestration platforms are used to provision and manage them.



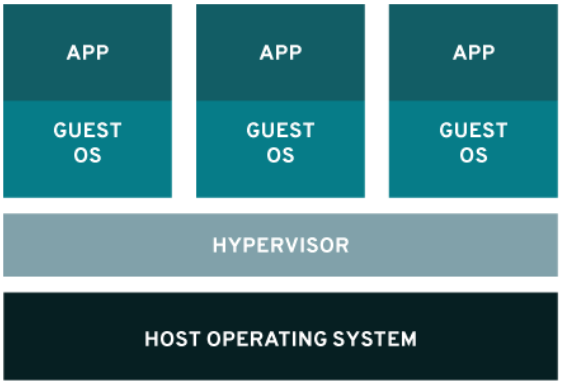
Virtual Machines are typically measured by the gigabyte. They usually contain their own OS, allowing them to perform multiple resource-intensive functions at once. The increased resources available to VMs allow them to abstract, split, duplicate, and emulate entire servers, OSs, desktops, databases, and networks.

VMs are capable of running far more operations than a single container, which is why they are the traditional way monolithic workloads have been (and are still today) packaged. But that expanded functionality makes VMs far less portable because of their dependence on the OS, application, and libraries. VMs are best used to:

- House traditional, legacy, and monolithic workloads
- Isolate risky development cycles
- Provision infrastructural resources (such as networks, servers, and data)
- Run a different OS inside another OS (such as running Unix on Linux)

Working

Software called a hypervisor separates resources from their physical machines so they can be partitioned and dedicated to VMs. When a user issues a VM instruction that requires additional resources from the physical environment, the hypervisor relays the request to the physical system and caches the changes. VMs look and act like physical servers, which can multiply the drawbacks of application dependencies and large OS footprints—a footprint that's mostly not needed to run a single app or microservice.



Stages of Linux Boot Process

1. BIOS

- BIOS stands for Basic Input/Output System
- Performs some system integrity checks
- Searches, loads, and executes the boot loader program.
- It looks for boot loader in floppy, cd-rom, or hard drive. User can press a key (typically F12 of F2, but it depends on your system) during the BIOS startup to change the boot sequence.
- Once the boot loader program is detected and loaded into the memory, BIOS gives the control to it.
- So, in simple terms BIOS loads and executes the MBR boot loader.

2. MBR

- MBR stands for Master Boot Record.
- It is located in the 1st sector of the bootable disk.
- MBR is less than 512 bytes in size. This has three components 1) primary boot loader info in 1st 446 bytes 2) partition table info in next 64 bytes 3) mbr validation check in last 2 bytes.
- It contains information about GRUB (or LILO in old systems).
- So, in simple terms MBR loads and executes the GRUB boot loader.

3. GRUB

- GRUB stands for Grand Unified Bootloader.
- If there are multiple kernel images installed on the system, one of them can choosen to be executed.
- GRUB has the knowledge of the filesystem (the older Linux loader LILO didn't understand filesystem).
- So, in simple terms GRUB just loads and executes Kernel and initrd images.

4. Kernel

- Mounts the root file system as specified in the "root=" in grub.conf
- Kernel executes the /sbin/init program
- Since init was the 1st program to be executed by Linux Kernel, it has the process id (PID) of 1. Do a 'ps -ef | grep init' and check the pid.
- initrd stands for Initial RAM Disk.
- initrd is used by kernel as temporary root file system until kernel is booted and the real root file system is mounted. It also contains necessary drivers compiled inside, which helps it to access the hard drive partitions, and other hardware.

5. Init

- Looks at the /etc/inittab file to decide the Linux run level.
- Following are the available run levels
    - 0 – halt
    - 1 – Single user mode
    - 2 – Multiuser, without NFS
    - 3 – Full multiuser mode
    - 4 – unused
    - 5 – X11
    - 6 – reboot
- Init identifies the default initlevel from /etc/inittab and uses that to load all appropriate program.
- Execute 'grep initdefault /etc/inittab' on your system to identify the default run level
- If you want to get into trouble, you can set the default run level to 0 or 6. Since you know what 0 and 6 means, probably you might not do that.

6. Runlevel programs

- When the Linux system is booting up, various services get started. Those are the runlevel programs, executed from the run level directory as defined by user's run level.
- Depending on default init level setting, the system will execute the programs from one of the following directories.

    - Run level 0 – /etc/rc.d/rc0.d/
    - Run level 1 – /etc/rc.d/rc1.d/
    - Run level 2 – /etc/rc.d/rc2.d/
    - Run level 3 – /etc/rc.d/rc3.d/
    - Run level 4 – /etc/rc.d/rc4.d/
    - Run level 5 – /etc/rc.d/rc5.d/
    - Run level 6 – /etc/rc.d/rc6.d/
- Under the /etc/rc.d/rc*.d/ directories, there are programs which start with S and K.
- Programs starts with S are used during startup. S for startup.
- Programs starts with K are used during shutdown. K for kill.
- There are numbers right next to S and K in the program names. Those are the sequence number in which the programs should be started or killed.

Linux Command Line Interface (CLI)

The Command Line Interface (CLI), is a non-graphical, text-based interface to the computer system, where the user types in a command and the computer then executes it. The Terminal is the platform or the IDE that provides the command line interface (CLI) environment to the user.

The CLI terminal accepts the commands that the user types and passes to a shell. The shell then receives and interprets what the user has typed into the instructions that can be executed by the OS (Operating System). If the output is produced by the specific command, then this text is displayed in the terminal. If any of the problems with the commands are found, then some error message is displayed.

Graphical User Interface

- In **Graphical Mode (GUI),** user can have many shells open, and perform tasks on multiple/remote computers. After successfully logging in, user is taken to the OS desktop where installed applications can be used.
- **Non-graphical mode (CLI)** starts off with a text-based login, User is prompted for our username/ID - password. If the login is successful, then an execution shell is provided. **In command line interface or the CLI**, there are no windows present to move around.

Shell

A Shell provides an interface to the Linux system. It gathers input from user and executes commands based on that input. When a program finishes executing, it displays that program's output.

Shell is an environment in which commands, programs, and shell scripts can be run. There are different flavors of a shell. Each flavor of shell has its own set of recognized commands and functions.

Shell Prompt

The prompt, **$**, which is called the command prompt, is issued by the shell. While the prompt is displayed, command can be used. Shell reads input after pressing Enter. It determines the command to be executed by looking at the first word of input.

Shell Types

There are two types of shells −

Bourne shell – In Bourne-type shell, the $ character is the default prompt.

It has following subcategories −

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

C shell − InC-type shell, the % character is the default prompt.

It has following subcategories −

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

The original Unix shell was written in the mid-1970s by Stephen R. Bourne while he was at the AT&T Bell Labs in New Jersey.

Bourne shell was the first shell to appear on Unix systems, thus it is referred to as "the shell".

Bourne shell is usually installed as /bin/sh on most versions of Unix. For this reason, it is the shell of choice for writing scripts that can be used on different versions of Unix.

Linux user

Users are accounts that can be used to login into a system. Each user is identified by a unique identification number or UID by the system. All the information of users in a system are stored in /etc/passwd file. The hashed passwords for users are stored in /etc/shadow file.

Users can be divided into two categories on the basis of the level of access:

Super user/root/administrator: Access to all the files on the system.

Normal users: Limited access.

When a new user is created, by default system takes following actions:

- Assigns UID to the user.
- Creates a home directory /home/.
- Sets the default shell of the user to be /bin/sh.
- Creates a private user group, named after the username itself.
- Contents of /etc/skel are copied to the home directory of the new user.
- .bashrc, .bash_profile and .bash_logout are copied to the home directory of new user. These files provide environment variables for this user's session.

Chapter 3 File system

- *A file* is a named collection of related information that is recorded on secondary storage.
- The file name is usually a string of characters, such as *example.c.*
- Information in a file is defined by its creator.
- Many different types of information may be stored in a file-source programs, object programs, executable programs, numeric data, text, payroll records, graphic images, sound recordings, and so on.
- *A* file has a certain defined *Structure* which depends on its type.
- Example:
  - *A text* file is a sequence of characters organized into lines (and possibly pages).
  - *A source* file is a sequence of subroutines and functions, each of which is further organized as declarations followed by executable statements.
  - An *object* file is a sequence of bytes organized into blocks understandable by the system's linker.
  - An *executable* file is a series of code sections that the loader can bring into memory and execute.

Path names

Path names can be of two types: **absolute** and **relative**.

An **absolute path** name begins at the root and follows a path down to the specified file, giving the directory names on the path.

A **relative path** name defines a path from the current directory.

Navigating the file system

Navigating the file system will helps in moving through the file system. Several commands are provided by Linux/Windows operating systems to navigate the file system. The commands to move from one directory to other, list the files available, copy/create/delete files from one place to other, create/ remove directories and many such operations are provided.

File types

A file is a named collection of related information that is recorded on secondary storage.

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes com-pressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

File Attributes

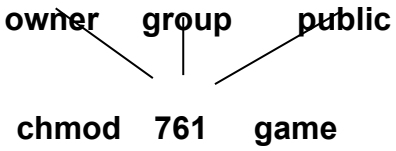
- **Name.** The symbolic file name is the only information kept in human-readable form.
- **Identifier.** This unique tag, usually a number, identifies the file within the file system; it is the non-human-readable name for the file.
- **Type.** This information is needed for systems that support different types of files.
- **Location.** This information is a pointer to a device and to the location of the file on that device.
- **Size.** The current size of the file (in bytes, words, or blocks) and possibly the maximum allowed size are included in this attribute.
- **Protection.** Access-control information determines who can do reading, writing, executing, and so on.
- **Time, date, and user identification.** This information may be kept for creation, last modification, and last use. These data can be useful for protection, security, and usage monitoring.

Information about files are kept in the directory structure, which is maintained on the disk

Access Control List (ACL)

- The common approach to the protection problem is to make access dependent on the identity of the user.
- An Access Control List (ACL) specifies user names and the types of access allowed for each user.
- When a user requests access to a particular file, the operating system checks the access list associated with that file. if the user has qualifications to access the file in the mode requested. The request is either allowed or denied.
- The modes of access are: read, write and execute.
- The main disadvantage of Access List is its length. As we do not know the users of the resources in advance, and the present directory entry is of varying size leading space management.
    - *This is solved by having three classifications of users.*
    - *Owner. The user who created the file is the owner.*
    - *Group. A set of users who are sharing the file and need similar access is a group, or work group.*
    - *Universe. All other users in the system constitute the universe.*
- Example: in UNIX System:
- Three classes of users

			RWX
a) owner access	7	⇒	1 1 1
			RWX
b) group access	6	⇒	1 1 0
			RWX
c) Universe public access	1	⇒	0 0 1



Adding text to file

Text can be added to a file in different ways in Linux operating system:

1. Using editors : vi editor, VIM, Nano, GEdit, Text Editor etc
2. Using redirection :
   - The operator > writes output to a new file, or to an existing file; in which case it wipes off everything which is previously present in the file.
   - The operator >> appends output at the end of an existing file.
   - E.g.: echo 'text here' >> filename

Pipes

Pipes are useful to connect multiple commands together. Instead of sending the output of a process to a file with redirect operator ">", user can pass it to another process as input using the pipe operator "|".
E.g.: ls –l | less.
In this example the output of the ls command is feed into less.

File Comparison

The file comparison commands compare the files and find the similarities and differences between these files. The different file comparison commands used in Linux are: cmp, comm, diff, uniq.

comm: Displays what is common between both the files
Syntax: comm file1 file2 //The contents of input files should be sorted alphabetically.
E.g.: Create 2 files with name f1 and f2
Contents of file f1: apple, mango, orange
Contents of file f2: grape, mango, musk melon
Example: comm f1 f2

apple
        grape
                mango
        muskmelon
orange

Column 1 gives the names which are present in first file but not in second file
Column 2 gives the names which are present in second file but not present in first file
Column 3 gives the names which are present in both the Files

diff: Display file differences. Diff displays which lines in one file have to be changed to make the file identical to other.
Syntax: diff file1 file2

E.g.:
Contents of file f1: apple, mango, orange
Contents of file f2: grape, mango, musk melon
diff f1 f2

Understanding Output

c – change d – delete a – append
1c1
<apple…
grape>

1c1: 1st line of 1st file to be changed to match 1st line of 2nd file
"<" means 1st file
">" means 2nd file
3c3: likewise can be understood.

Filters / Text Processing Commands

Filters are commands which accept data from standard input manipulate it and write the results to standard

Output.
head: command displays the top "n" lines of the file. When used without any option it will display first 10 lines of the file. (Note: Create a file sample.txt using text editor and type at least 15 lines)
Syntax: head filename
Example: head sample1.txt
/*display first 10 lines*/

tail: command displays the end of the file. By default it will display last 10 lines of the file.
Syntax: tail filename
Example: head sample1.txt
/*display last 10 lines*/
(Note: Create a file sample.txt using text editor and type at least 15 lines)
tail or head with -n followed by a number will display that many number of lines from last and from first respectively.
head -n 20 sample1.txt
/* will display first 20 lines*/
tail -n 15 sample1.txt
/* will display last 15 lines */

cut: Cut command in linux is used to select sections of text from each line of files. The cut command can be used to select fields or columns from a line by specifying a delimiter or to select a portion of text by specifying the range or characters. Basically the cut command slices a line and extracts the text.

Syntax: cut -c [numbers delimited by comma or range] <file name>
Here the option available are c : character
f : field
d : delimeter

Example: Create a file using text editor "file.txt" with following data
unix or linux os
is unix good os
is linux good os

cut -c4 file.txt

Displays 4th character from the file

x
u
l

cut -c4,6 file.txt
Displays 4th & 6th characters from the file

xo
ui
ln

cut -c4-7 file.txt

The above cut command prints the characters from fourth position to the seventh position in each line.

x or
unix

linu

The cut command can be used to extract the fields in a file using a delimiter. The -d option in cut command can be used to specify the delimiter and -f option is used to specify the field position.
cut -d' ' -f2 file.txt
or
unix
linux

This command prints the second field in each line by treating the space as delimiter.
More than one field can be printed by specifying the position of the fields in a comma delimited list.

cut -d' ' -f2,3 file.txt

or linux
unix good
linux good

The above command prints the second and third field in each line.
User can print a range of fields by specifying the start and end position.
cut -d' ' -f1-3 file.txt

The above command prints the first, second and third fields.

To print the first three fields, you can ignore the start position and specify only the end position.
cut -d' ' -f-3 file.txt

paste:
Paste command will paste the contents of the files side by side
create “file1” with following contents using text editor and contents must be separated by TAB

sourav	40	male
sachin	45	male
pooja	16	female
suchithra	17	female

create “file2” with following contents using text editor and contents must be separated by TAB

IV	CS	NRAMP
II	EC	KARKALA
VI	CE	MANIPAL
II	EE	UDUPI

Syntax: paste file1 file2
Example: paste cutlist1.txt cutlist2.txt

sourav	40	male	IV	CS	NRAMP
sachin	45	male	II	EC	KARKALA
pooja	16	female	VI	CE	MANIPAL
suchithra	17	female	II	EE	UDUPI

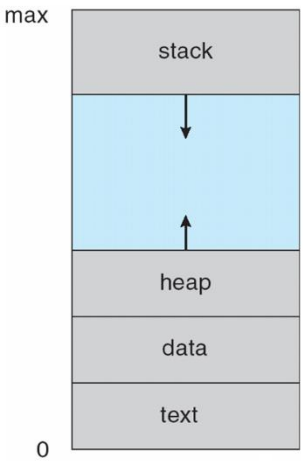
sort: Sorts the lines in a text file.
create a “FRUITS” with following contents using text editor
mango
apple

orange
banana
grapes
Syntax: sort [options]... [file]

Chapter 4 Process Management

Process

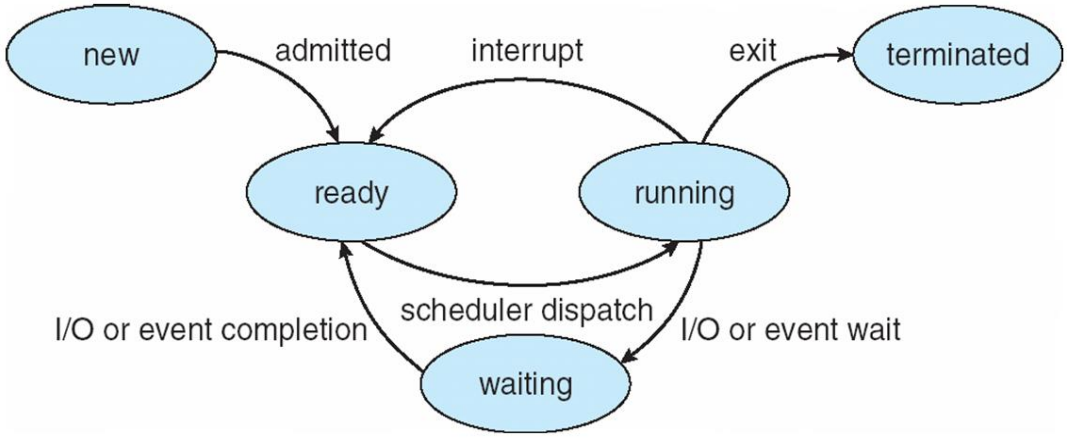
A process is a program in execution. A process is more than the program code, which is sometimes known as the text section. It also includes the current activity, as represented by the value of the program counter and the contents of the processor's registers. A process generally also includes the process stack, which contains temporary data (such as function parameters, return addresses, and local variables), and a data section, which contains global variables. A process may also include a heap, which is memory that is dynamically allocated during process run timeProcess is an active entity



Foreground processes interact with users. Background processes that stay in background sleeping but suddenly springing to life to handle activity such as email, webpage, printing, and so on.

Background processes are called daemons.

Process states



As a rrent activity of that process. Each process may be in one of the following states:

- **New.** The process is being created.
- **Running.** Instructions are being executed.
- **Waiting.** The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
- **Ready.** The process is waiting to be assigned to a processor.
- **Terminated.** The process has finished execution.

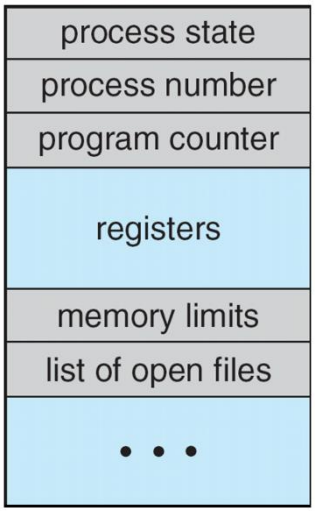
Process Control Block (PCB)

Process Control Block is the way of representing the information about each process in an operating system. It is also called a **Task Control Block (TCB)**.

Contents of PCB are:

- **Process state:** Specifies the current state of the process. The states may be new, ready, running, waiting, and terminated.
- **Program counter:** Address of the next instruction to be executed for this process.

- **CPU registers:** Memory locations which is direct accessible to CPU. These register vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, general purpose registers, and any condition-code information.
- **CPU scheduling information:** This includes a process priority, pointers to scheduling queues, and any other scheduling parameters.
- **Memory-management information:** Includes the information to manage the memory for the execution of the process. This includes value of base and limit register, the page tables, or the segment tables, depending on the memory system used by the OS.
- **Accounting information:** This includes the amount of CPU and real time used, time limits, account numbers, job or process numbers and so on.
- **I/O status information:** This includes the list of I/O devices allocated to the process, a list of open files, and so on.

Figure: 

CB)

Process scheduling Queue

Scheduling queues refers to queues of processes or devices. When the process enters into the system, then this process is put into a job queue. This queue consists of all processes in the system.

The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the **ready queue**.

Device queue is a queue for which multiple processes are waiting for a particular I/O device. Each device has its own device queue.

Queues are generally stored as a linked list.

This figure shows the queuing diagram of process scheduling.

- Queue is represented by rectangular box.
- The circles represent the resources that serve the queues.
- The arrows indicate the process flow in the system.

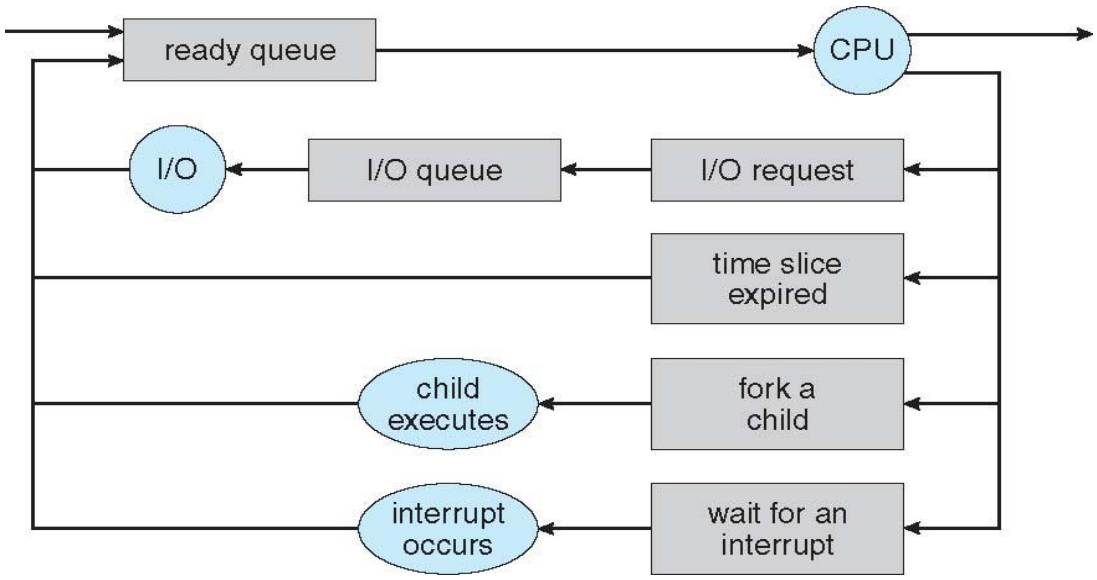


Figure: Queuing-Diagram representation of Process Scheduling.

Queues are of two types

- Ready queue
- Device queue

A newly arrived process is put in the ready queue. Processes waits in ready queue for allocating the CPU. Once the CPU is assigned to a process, then that process will execute. While executing the process, any one of the following events can occur.

1. The process could issue an I/O request and then it would be placed in an I/O queue.
2. The process could create new sub process and will wait for its termination.
3. The process could be removed forcibly from the CPU, as a result of interrupt and put back in the ready queue.

Operations on Processes

Process Creation:

A process may create several new processes, via a *create-process system call*, during the course of execution. The creating process is called a *parent process*, and the new processes are called the *children* of that process. Each of these new processes may in turn create other processes, forming a tree of processes.

When a process creates a new process, two possibilities exist in terms of execution:

- The parent continues to execute concurrently with its children.
- The parent waits until some or all of its children have terminated.

There are also two possibilities in terms of the address space of the new process:

- The child process is a duplicate of the parent process (it has the same program and data as the parent).
- The child process has a new program loaded into it.

Example: Creating separate process using UNIX fork( ) System Call

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

```
        if( pid = = 0)
        {
                printf("Child Process Created");
        }
        else
        {
                printf("Paraent Process Created");
        }
        return 0;
}
```

Process Termination:

A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the exit ( ) system call.

A process can cause the termination of another process via an appropriate system call. Usually, such a system call can be invoked only by the parent of the process that is to be terminated. Otherwise, users could arbitrarily kill each other's jobs.

A parent may terminate the execution of one of its children for a variety of reasons, such as these:

1. The child has exceeded its usage of some of the resources that it has been allocated. (To determine whether this has occurred, the parent must have a mechanism to inspect the state of its children.)
2. The task assigned to the child is no longer required.
3. The parent is exiting, and the operating system does not allow a child to continue if its parent terminates.

In some systems, if a process terminates (either normally or abnormally), then all its children must also be terminated. This phenomenon, referred to as *cascading termination*, is normally initiated by the operating system.

Interprocess communication

Cooperating Processes:

Any process that shares data with other processes is a *cooperating process*. Cooperating processes require an interprocess communication (IPC) mechanism that will allow them to exchange data and information.

There are two fundamental models of interprocess communication:

1. Shared memory
2. Message passing.

Shared Memory	Message Passing
Processes exchange information by reading and writing data to the *shared region*.	Communication between processes takes place by *exchanging messages*.
Large amounts of data can be exchanged.	Useful for exchanging *smaller* amounts of data
It is *difficult* to implement.	It is *easier* to Implement.
Here communication is *fast*. Because it does not requires the assistance of kernel except for the establishment of shared-memory.	Here communication is *slow*. Because as it uses system call, it requires the assistance of kernel

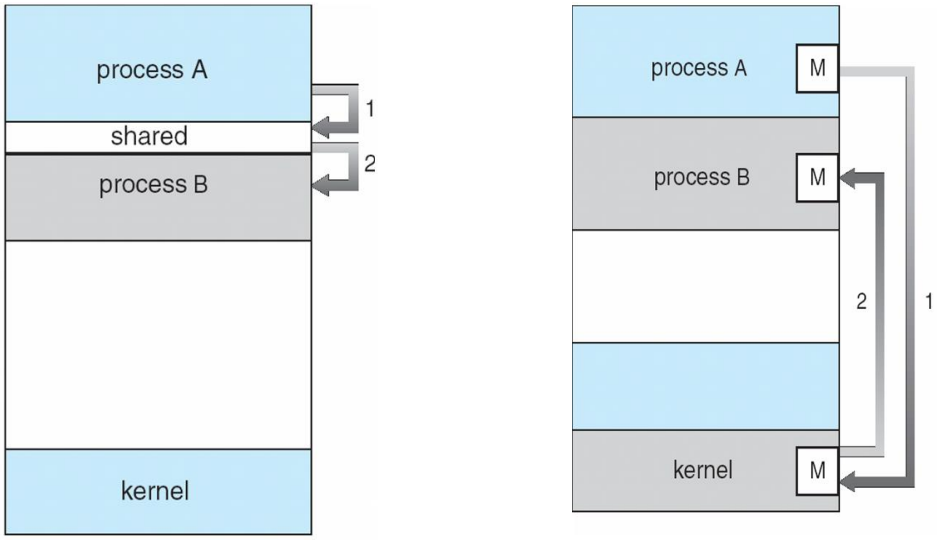


Figure: Shared Memory Message Passing

Scheduling

Process Scheduling: The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular **strategy.**

The following are the different type of schedulers

1. **Long-term scheduler (or job scheduler) –** selects processes from job pool on disk and brings them into memory for execution. Long-term scheduler is invoked very infrequently (seconds, minutes). The long-term scheduler controls the *degree of multiprogramming.*
2. **Short-term scheduler (or CPU scheduler) –** selects a process from ready queue to the processor for execution. Short-term scheduler is invoked very frequently (milliseconds). (Must be fast).
3. **Medium-term schedulers:** Some time it is required to remove partially executed process from memory and later reintroduce into memory to continue its execution. This is called **SWAPPING** and this is done by medium term scheduler.

Context Switch:

"The mechanism of Switching the CPU to another process by saving the current state and restoring another process is known as context switch".

1. When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process.
2. Context-switch time is overhead; the system does no useful work while switching.
3. Time dependent on hardware support

Different types of CPU schedulers

The assignment of physical processors to processes allows processors to accomplish work. The problem of determining when processors should be assigned and to which processes is called processor scheduling or CPU scheduling. When more than one process is runnable, the operating system must decide which one first. The part of the operating system concerned with this decision is called the scheduler, and algorithm it uses is called the scheduling algorithm.

CPU Scheduler

a. Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

b. CPU scheduling decisions may take place when a process:

1. Switches from running to waiting state
2. Switches from running to ready state
3. Switches from waiting to ready
4. Terminates

✓ Scheduling under 1 and 4 is ***nonpreemptive***

✓ All other scheduling is ***preemptive***

Nonpreemptive processes are those which cannot be removed from the processor until they finish their execution.

Preemptive processes are those which can be removed (interrupted) from the processor before they complete their execution and can be rescheduled later for execution.

Dispatcher

Dispatcher module gives control of the CPU to the process selected by the **short-term scheduler**; this involves:

- switching context
- switching to user mode
- jumping to the proper location in the user program to restart that program

Dispatch latency – time it takes for the dispatcher to stop one process and start another running.

CPU Scheduling Criteria:

- **CPU utilization.** The CPU should be *kept busy* as much as possible. *Maximize CPU utilization*.
- **Throughput.** The number of processes that are completed per time unit, called *throughput*. *Maximize Throughput*.
- **Turnaround time.** The interval from the time of submission of a process to the time of completion is the *turnaround time*. Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O. *Minimize Turnaround time*.
- **Waiting time.** The CPU-scheduling algorithm does affects only the amount of time that a process spends waiting in the ready queue. Waiting time is the sum of the periods spent waiting in the ready queue. *Minimize Waiting time*.
- **Response time.** The time from the submission of a request until the first response is produced is called *response time*, is the time it takes to start responding, not the time it takes to output the response. The turnaround time is generally limited by the speed of the output device. *Minimize Response time*.

Scheduling Algorithms:

- First Come First Serve (FCFS) Scheduling
- Shortest-Job-First (SJF) Scheduling
- Priority Scheduling
- Round Robin(RR) Scheduling
- Multilevel Queue Scheduling
- Multilevel Feedback Queue Scheduling

==========

First Come First Serve FCFS

- Jobs are executed on first come, first serve basis.
- Easy to understand and implement (Write).
- Poor in performance as average wait time is high. That is average waiting time is not minimal and ma vary substantially if the processes CPU burst times vary greatly.

Example: Processes arrive at time 0,

Process	**Burst Time(in Milliseconds)**
P1	24
P2	3
P3	3

Assume arriving order of processes : P1, P2, P3

Gantt Chart:

Average Waiting Time is (0 + 24 + 27) / 3 = 27milliseconds.

Assume arriving order of processes: P2, P3, P1

Gantt Chart:

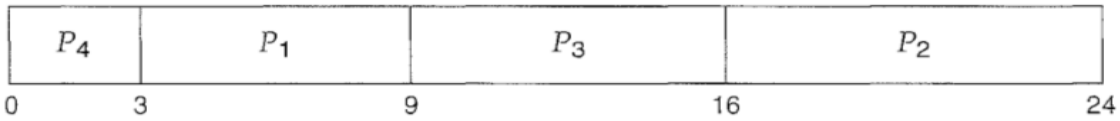
Average Waiting Time is (0 + 3 + 6) / 3 = 3milliseconds

Shortest Job First (SJF)

- Associates with each process the length of the process's next CPU burst. When the CPU is available, it is assigned to the process that has the smallest next CPU burst. If the next CPU bursts of two processes are the same, FCFS scheduling is used to break the tie.
- Best approach to minimize waiting time.(provably optimal)
- Impossible to implement
- Processor should know in advance how much time process will take.
- It can be either Pre-emptive or Non-pre-emptive.(Based on Burst Time)
- It is sometimes called *Shortest Remaining Time First(SRTF)* Scheduling

Process	**Burst Time (in milliseconds)**
P1	6
P2	8
P3	7
P4	3

Gantt Chart:



Average Waiting Time: for processes P4, P1, P3, P2

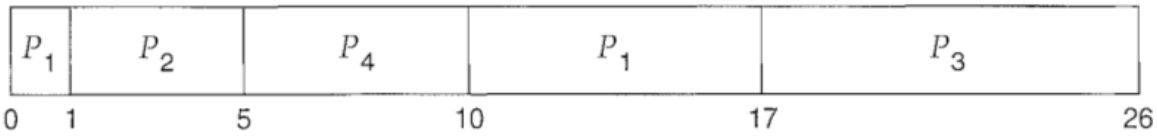
(0 + 3 + 9 + 16) / 4 = 7milliseconds.

==========

Example for SJF (SRTF) Scheduling:

Process	**Arrival Time**	**Burst Time (Milliseconds)**
P1	0	8
P2	1	4
P3	2	9
P4	3	5

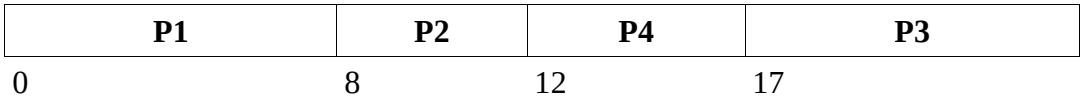
Gantt Chart: For Pre-emptive SJF Scheduling Algorithm



Average Waiting Time: For **Pre-emptive** SJF Scheduling Algorithm

[(10 – 1) + (1 – 1) + (17 – 2) + (5 – 3)] / 4 = 26 / 4 = 6.5 milliseconds.

Gantt Chart: For Non-pre-emptive SJF Scheduling Algorithm



Average Waiting Time: For **Non-pre-emptive** SJF Scheduling Algorithm

[(0) + (8 – 1) + (17 – 2) + (12 – 3)] / 4 = 31 / 4 = 7.75 milliseconds.

Process priority (priority scheduling algorithm)

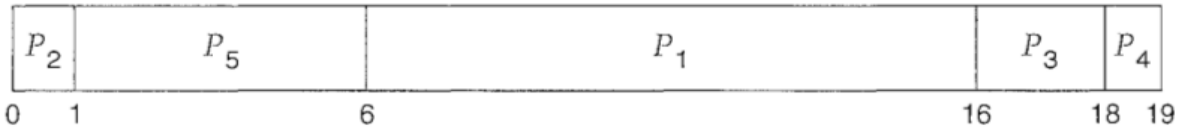
Each process is assigned a priority. Process with highest priority is to be executed first and so on.

- Processes with same priority are executed on first come first serve basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.
- It can be either Pre-emptive or Non-pre-emptive. (Based on Priority)
- Low priority processes may face indefinite blocking – called **Starvation**. Solution to this problem is **Aging**. Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time.

Example:

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Gantt Chart:



Average Waiting Time for the processes: P2, P5, P1, P3, P4

(0 + 1 + 6 + 16 + 18) / 5 = 41 / 5 = 8.2 milliseconds.

Assigning priorities to processes:

Priorities to the processes can be defined either internally or externally.

- **Internally** defined priorities use some measurable quantity or quantities to compute the priority of a process. Example, time limits, memory requirements, the number of open files, and the ratio of average I/O burst to average CPU burst have been used in computing priorities.
- **External priorities** are set by criteria outside the operating system, such as the importance of the process, the type and amount of funds being paid for computer use, the department sponsoring the work, and other, often political factors.

Debugging

Debugging is the process of detecting and removing of existing and potential errors (also called as 'bugs') in a software code that can cause it to behave unexpectedly or crash. To prevent incorrect operation of a software or system, debugging is used to find and resolve bugs or defects. When various subsystems or modules are tightly coupled, debugging becomes harder as any change in one module may cause more bugs to appear in another. Sometimes it takes more time to debug a program than to code it.

Chapter 5 Process synchronization

Critical section problem

Critical Section: Is a code segment that accesses shared variables and has to be executed as an atomic action.

Critical Section Problem: The problem of how to ensure that at most one process is executing its critical section at a given time.

A solution to the critical-section problem must satisfy the following three requirements:

- **Mutual exclusion**. If process Pi is executing in its critical section, then no other processes can be executing in their critical sections.
- **Progress.** If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can participate in deciding which will enter its critical section next, and this selection cannot be postponed indefinitely.
- **Bounded waiting.** There exists a bound, or limit, on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

Semaphores

A *synchronization tool* used to solve critical-section problem is called **Semaphore**. It is first proposed by *E. Dijkstra.*

A semaphore S is an *integer variable* that, apart from initialization, is accessed only through two standard *atomic operations*: **wait ( )** and **signal ( )**.

The wait ( ) operation was originally termed P (from the Dutch *Proberen*, "to test"); signal( ) was originally called V (from *Verhogen*, "to increment").

The definition of ***wait ( )*** is as follows:

```
wait ( S ) {
        while ( S <= 0 )
                ; // no-op
        S – –
}
```

The definition of ***signal( )*** is as follows:

```
signal ( S ) {
        S++;
}
```

- As wait ( ) and signal ( ) are atomic operations, when one process modifies the semaphore value, no other process can simultaneously modify that same semaphore value.
- In the case of wait (S), the testing of the integer value of S (S <= 0), as well as its possible modification (S – –), must be executed **without interruption**.

Deadlock

"When processes request a resource and if the resources are not available at that time the process enters into waiting state. Waiting process may not change its state because the resources they are requested are held by other process. This situation is called **deadlock".**

System model

A system may consist of finite number of resources and is distributed among number of processes. These resources are partitioned into several instances each with identical instances.

A process must request a resource before using it and it must release the resource after using it. It can request any number of resources to carry out a designated task. The amount of resource requested may not exceed the total number of resources available.

A process may utilize the resources in only the following sequences:

1. **Request:** If the request is not granted immediately then the requesting process must wait until it can acquire the resources.
2. **Use:** The process can operate on the resource. i.e use the memory to perform its task.
3. **Release:**-The process releases the resource after using it.

Deadlock may involve different types of resources.
For eg:-Consider a system with **one printer** and **one tape drive**. If a process Pi currently holds a printer and a process Pj holds the tape drive. If process Pi request a tape drive and process Pj request a printer then a deadlock occurs.

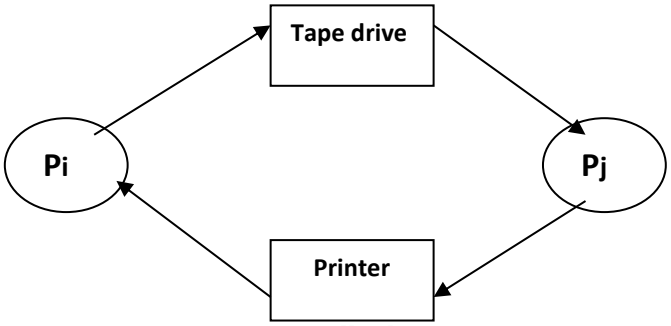


Fig: Deadlock situation

Methods for handling deadlocks

There are three ways to deal with deadlock problem

1. Use a protocol to prevent deadlocks ensuring that the system will never enter into the deadlock state.
2. Allow a system to enter into deadlock state, detect it and recover from it.
3. Ignore the problem and pretend that the deadlock never occur in the system.

Deadlock prevention

For a deadlock to occur, each of the four necessary conditions ***must*** hold. By ensuring that at least one of these conditions cannot hold, we can *prevent* the occurrence of a deadlock. The necessary four conditions are: *Mutual Exclusion, Hold and Wait, No pre-emption, Circular Wait*.

Mutual Exclusion:

The mutual-exclusion condition must hold for non-sharable resources (like printers). Sharable resources (like read-only files), do not require mutually exclusive access and thus cannot be involved in a deadlock. In general, however, we cannot prevent deadlocks by denying the mutual-exclusion condition, because *some resources are intrinsically nonsharable*.

Hold and Wait:

To ensure that the hold-and-wait condition never occurs in the system, we must guarantee that, whenever a process requests a resource, it does not hold any other resources.

Protocols used are;

- Requires each process to request and be allocated all its resources before it begins execution.
- Allows a process to request resources only when it has none.

Disadvantages of the above protocols are;

1. Resource utilization may be low.
2. Starvation is possible.

No preemption:

No preemption does not holds in the system can be ensured by the following protocol.

- If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released. (i.e. implicitly released, only if another process requests them.)
- Pre-empted resources are added to the list of resources for which the process is waiting

- Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting

Circular Wait:

One way to ensure that *circular wait* condition never holds is to impose a total ordering of all resource types and to require that each process requests resources in an increasing order of enumeration.

Deadlock avoidance

- Deadlock prevention algorithm may lead to low device utilization and reduces system throughput.
- Avoiding deadlocks requires additional information about how resources are to be requested. With the knowledge of the complete sequences of requests and releases we can decide for each requests whether or not the process should wait.
- For each requests it requires to check the resources currently available, resources that are currently allocated to each processes future requests and release of each process to decide whether the current requests can be satisfied or must wait to avoid future possible deadlock.
- A deadlock avoidance algorithm dynamically examines the resources allocation state to ensure that a circular wait condition never exists. The resource allocation state is defined by the number of available and allocated resources and the maximum demand of each process.

Safe State:

"A state is safe state in which there exists at least one order in which all the process will run completely without resulting in a deadlock".

- A system is in safe state if there exists a safe sequence.
- A sequence of processes <P1,P2,……..Pn> is a safe sequence for the current allocation state if for each Pi
  I. the resources that Pi can request can be satisfied by the currently available resources.
  II. If the resources that Pi requests are not currently available then Pi can obtain all of its needed resource to complete its designated task.
- A safe state is not a deadlock state.
- Whenever a process request a resource i.e., currently available, the system must decide whether resources can be allocated immediately or whether the process must wait. The request is granted only if the allocation leaves the system in safe state.
- In this, if a process requests a resource i.e., currently available it must still have to wait. Thus resource utilization may be lower than it would be without a deadlock avoidance algorithm.

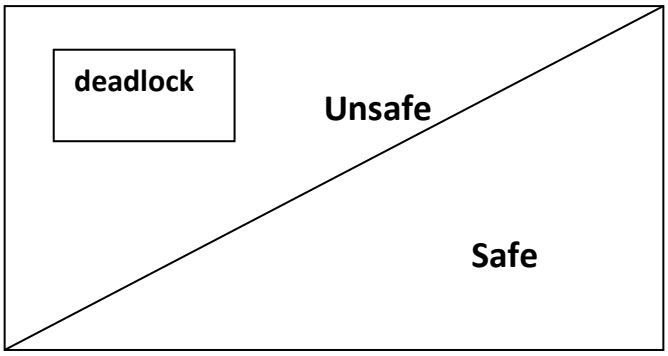


Fig: process safe, unsafe and deadlock state.

Resource Allocation Graph Algorithm:

- ✓ This algorithm is used only if we have one instance of a resource type. In addition to the **request edge** and the **assignment edge** a new edge called **claim edge** is used. *For eg*:-A claim edge Pi-→Rj indicates that process Pi may request Rj in future. The claim edge is represented by a dotted line.
- ✓ When a process Pi requests the resource Rj, the claim edge is converted to a request edge.
- ✓ When resource Rj is released by process Pi, the assignment edge Rj→Pi is replaced by the claim edge Pi→Rj.

- ✓ When a process Pi requests resource Rj the request is granted only if converting the request edge Pi→Rj to as assignment edge Rj→Pi do not result in a cycle. Cycle detection algorithm is used to detect the cycle. If there are no cycles then the allocation of the resource to process leave the system in safe state.

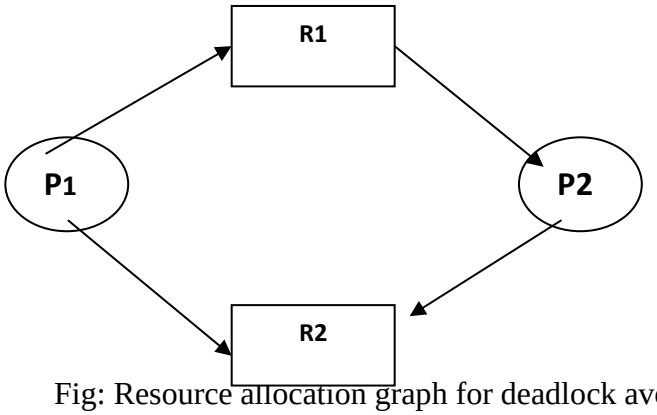


Fig: Resource allocation graph for deadlock avoidance

Here P1 is requesting for R1 and R2, as R1 is allocated to P2. Also P2 is requesting to R2,but at this time R2 should be allocated to P1 to avoid the cycle formation. Here P2 already holding R1 hence until release of R1 by P2 we cannot allocate R2 to P2.

Banker's Algorithm (Dijkstra)

This algorithm is applicable to the system with multiple instances of each resource types, but this is less efficient then the resource allocation graph algorithm.

When a new process enters the system it must declare the maximum number of resources that it may need. This number may not exceed the total number of resources in the system. The system must determine that whether the allocation of the resources will leave the system in a safe state or not. If it is so resources are allocated else it should wait until the process release enough resources.

Several data structures are used to implement the banker's algorithm. Let 'n' be the number of processes in the system and 'm' be the number of resources types. We need the following data structures:

i. **Available**:-A vector of length **m** indicates the number of available resources.
If Available[i]=k, then k instances of resource type Rj is available.

ii. **Maximum**:- An n*m matrix defines the maximum demand of each process.
if Max[i,j]=k, then Pi may request at most k instances of resource type Rj.

iii.**Allocation**:-An n*m matrix defines the number of resources of each type currently allocated to each process. If Allocation[i,j]=k, then Pi is currently k instances of resource type Rj.

iv.**Need**:-An n*m matrix indicates the remaining resources need of each process. If Need[i,j]=k, then Pi may need k more instances of resource type Rj to compute its task.

So **Need[i,j]=Max[i,j]-Allocation[i,j]**

Safety Algorithm:

This algorithm is used to find out whether or not a system is in safe state or not.

Step 1. Let work and finish be two vectors of length M and N respectively. Initialize work = available and

Finish[i]=false for **i=1,2,3,…….n**

Step 2. Find i such that both Finish[i]=false & Need i <= work
If no such i exist then go to step 4

Step 3. Work = work + Allocation , Finish[i]=true, Go to step 2

Step 4 . If finish[i]=true for all i, then the system is in safe state. This algorithm may require an order of m*n*n operation to decide whether a state is safe.

Resource Request Algorithm: Let Request(i) be the request vector of process Pi. If Request(i)[j]=k, then process Pi wants K instances of the resource type Rj. When a request for resources is made by process Pi the following actions are taken.

- ✓ If Request(i) <= Need(i) go to step 2 otherwise raise an error condition since the process has exceeded its maximum claim.
- ✓ If Request(i) <= Available go to step 3 otherwise Pi must wait. Since the resources are not available.
- ✓ If the system want to allocate the requested resources to process Pi then modify the state as follows.

  Available = Available – Request(i)
  Allocation(i) = Allocation(i) + Request(i)
  Need(i) = Need(i) – Request(i)

- ✓ If the resulting resource allocation state is safe, the transaction is complete and Pi is allocated its resources. If the new state is unsafe then Pi must wait for Request(i) and old resource allocation state is restored.

Recovery from deadlock

When a detection algorithm determines that a deadlock exists, alternatives available for the recovery of the deadlock are;

Manual Recovery: Inform the operator that a deadlock has occurred and to let the operator deal with the deadlock manually.

Automatic Recovery: System itself recover from deadlock automatically. Here two ways are there to break the deadlock.

1. **Process Termination:** One is simply to abort one or more processes to break the circular wait.
2. **Resource Preemption:** Preempt some resources from one or more of the deadlocked processes.

Process Termination

Two methods for process termination to remove deadlock;

- **Abort all deadlocked processes**. This method *clearly will break the deadlock cycle*, but at *great expense*; the deadlocked processes may have computed for a long time, and the results of these partial computations must be discarded and probably will have to be recomputed later; i.e. a lot of process work is lost.
- **Abort one process at a time until the deadlock cycle is eliminated**. This method incurs considerable *overhead*, since after each process is aborted, a deadlock-detection algorithm must be invoked to determine whether any processes are still deadlocked.

In both methods, the system reclaims all resources allocated to the terminated processes.

Resource Pre-emption:

To eliminate deadlocks using resource pre-emption, we successively pre-empt some resources from processes and give these resources to other processes until the deadlock cycle is broken.

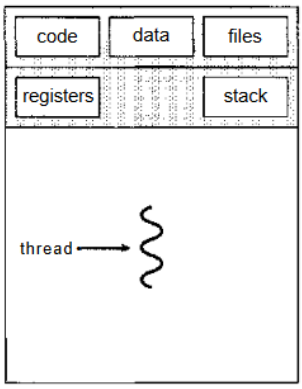
If pre-emption is required to deal with deadlocks, then three issues need to be addressed:

1. **Selecting a victim.** Which resources and which processes are to be pre-empted?
2. **Rollback.** If we pre-empt a resource from a process, what should be done with that process? Roll back the process to some safe state and restart it from that state. Finding the safe state is a tedious task.
3. **Starvation.** How do we ensure that starvation will not occur? That is, how can we guarantee that resources will not always be pre-empted from the same process?

Victim selection is based on the cost factor. I.e. The same process is not picked as a victim. If so, that process never completes its task, leads to starvation. A process can be picked as a victim only a small (finite) number of times. (Example, number of roll backs a process can go through)

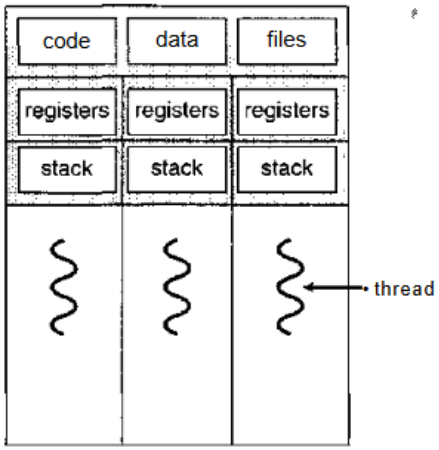
Threads

A thread is a basic unit of CPU utilization. It comprises a thread ID, a program counter, a register set, and a stack. A traditional (or heavyweight) process has a single thread of control



single-threaded process

If a process has multiple threads of control, it can perform more than one task at a time.



multithreaded process

Benefits of multithreaded programming:

1. Responsiveness.
2. Resource sharing.
3. Economy.
4. Utilization of multiprocessor architectures.

Multithreading Models

There are two types of threads:

User threads: are supported above the kernel and are managed without kernel support. User threads can be easily implemented and is implemented by the user. If a user performs a user-level thread blocking operation, the whole process is blocked. The kernel level thread does not know nothing about the user level thread.

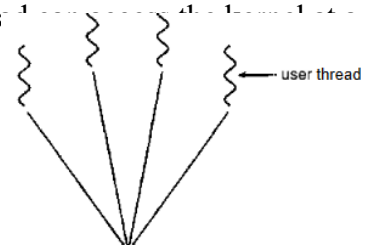
Kernel threads: are supported and managed directly by the operating system. The kernel thread recognizes the operating system. There is a thread control block and process control block in the system for each thread and process in the kernel-level thread. The kernel-level thread is implemented by the operating system. The kernel knows about all the threads and manages them. The kernel-level thread offers a system call to create and manage the threads from user-space. The implementation of kernel threads is more difficult than the user thread. Context switch time is longer in the kernel thread.

There must exist a relationship between user threads and kernel threads

Many-to-One Model

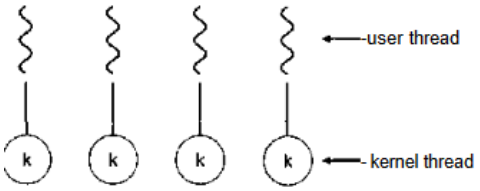
Maps many user-level threads to one kernel thread. Thread management is done by the thread library in user

space, so it is efficient; but the entire process will block if a thread makes a blocking system call. Also, because only one thre[illegible] time, multiple threads are unable to run in parallel on multiprocessors.



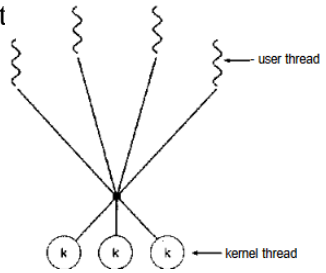
One-to-One Model

Maps each user thread to a kernel thread. It provides more concurrency than the many-to-one model by allowing another thread to run when a thread makes a blocking system call; it also allows multiple threads to run in parallel on multiprocessors. The only drawback to this model is that creating a user thread requires creating the corresponding kernel thread. Because the overhead of creating kernel threads can burden the performance of an application, most implementations of this model restrict the number of threads supported by the system. Linux, along with the family of Windows operating systems—including Windows 95, 98, NT, 2000, and XP— implement the one-to-one model.



Many-to-Many Model

The many-to-many model multiplexes many user-level threads to a smaller or equal number of kernel threads. The number of kernel threads may be specific to either a particular application or a particular machine (an -user thread- kernel thread Many-to-many model. application may be allocated more kernel threads on a multiprocessor than on a uniprocessor). Whereas the many-to-one model allows the developer to create as many user threads as she wishes, true concurrency is not gained because the kernel can schedule only one thread at



Chapter 6 Memory management

Process address space

An address space is a range of valid addresses in memory that are available for a program or process. That is, it is the memory that a program or process can access. The memory can be either physical or virtual and is used for executing instructions and storing data.

There are three types of addresses used in a program before and after memory is allocated

Symbolic addresses: The addresses used in a source code. The variable names, constants, and instruction labels are the basic elements of the symbolic address space.

Relative addresses: At the time of compilation, a compiler converts symbolic addresses into relative addresses.

Physical addresses: The loader generates these addresses at the time when a program is loaded into main memory.

The set of all logical addresses generated by a program is referred to as a **logical address space**. The set of all physical addresses corresponding to these logical addresses is referred to as a **physical address space**.

The runtime mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device. MMU uses following mechanism to convert virtual address to physical address.

The value in the base register is added to every address generated by a user process, which is treated as offset at the time it is sent to memory. For example, if the base register value is 10000, then an attempt by the user to use address location 100 will be dynamically reallocated to location 10100.

The user program deals with virtual addresses; it never sees the real physical addresses.

Dynamic loading:

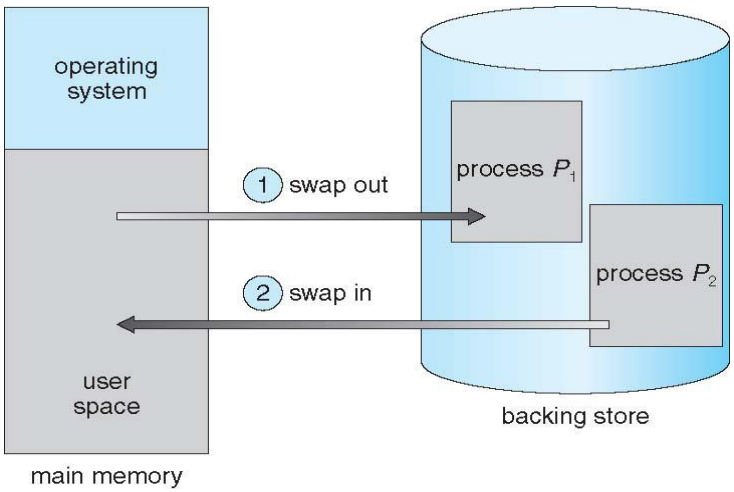
- Routine is not loaded until it is called
- Better memory-space utilization because unused routine is never loaded
- All routines kept on disk in relocatable load format
- It is useful when large amounts of code are needed to handle infrequently occurring cases
- No special support from the operating system is required
    - Implemented through program design
    - OS can help by providing libraries to implement dynamic loading

Dynamic Linking:

- *Static linking* – system libraries and program code combined by the loader into the binary program image
- *Dynamic linking* –linking postponed until execution time
- Small piece of code, *stub*, used to locate the appropriate memory-resident library routine
- Stub replaces itself with the address of the routine, and executes the routine
- Operating system checks if routine is in processes' memory address
    - If not in address space, add to address space
- Dynamic linking is particularly useful for libraries
- System also known as *shared libraries*
- Consider applicability to patching system libraries
    - Versioning may be needed

Swapping

- Here, a process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution.
- Total physical memory space of processes can exceed physical memory.



- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images.
- **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed.
- Major part of swap time is *transfer time*; total transfer time is directly proportional to the amount of memory swapped
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk
- Does the swapped out process need to swap back in to same physical addresses?
    - It depends on address binding method
    - Plus consider pending I/O to / from process memory space

Memory allocation

Divide memory into several fixed-sized partitions. Each partition may contain exactly one process. Thus, the degree of multiprogramming is bound by the number of partitions. In this multiple-partition method, when a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process.

Fragmentation

The memory which is wasted after allocating the memory to processes is called *fragmentation*.

There are two types of fragmentation.

4. *External Fragmentation:*
    - This happens where the system allocates memory in *Variable-sized* units.
    - Total memory space exists to satisfy a request, but it is not contiguous.
    - Solution to External Fragmentation: *Paging, Segmentation, and Compaction*:
        - *Compaction:* Shuffle memory contents to place all free memory together in one large block. Compaction is possible *only* if relocation is dynamic, and is done at execution time.
5. *Internal Fragmentation:*
    - This happens where the system allocates memory in *fixed-sized* units.
    - Allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.

Paging

- Paging is a memory-management scheme that permits the physical address space a process to be non-contiguous.
- Paging avoids external fragmentation and the need for compaction.
- It also solves the considerable problem of fitting memory chunks of varying sizes onto the backing store.

Implementation:

- Physical memory is divided into fixed-sized blocks called *Frames*. (size is power of 2, between 512 bytes to 16M bytes)
- Divides the logical memory in to blocks of same size called *Pages*.

- Backing Store is divided in to fixed-sized blocks (same size as the memory frames).

Paging Hardware:

- Every address generated by the CPU is divided into two parts: a *page number (p)* and a *page offset (d)*.
- Page Number (p) is used as an index into a page table. Page Table contains the base address (f) of each page in physical memory. This address (base address) is combined with page offset (d) to define the physical memory address.

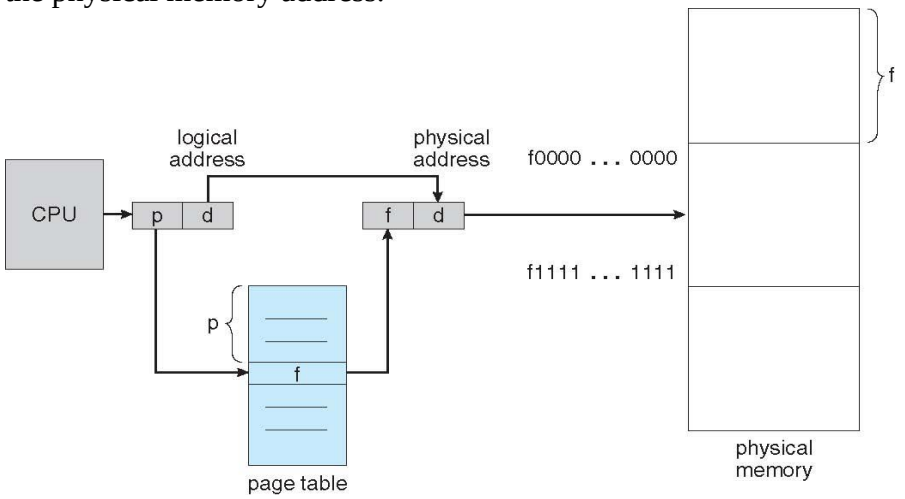


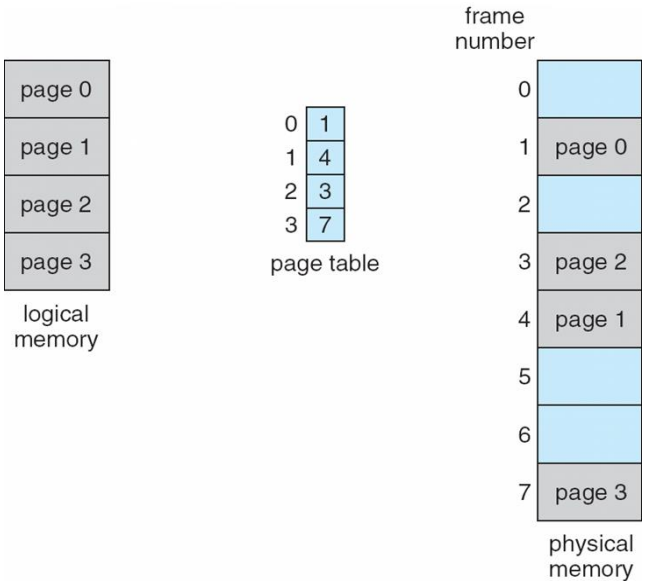
Figure: Paging Hardware

If the size of the logical address space is 2^m, and a page size is 2^n addressing units (bytes or words) then the high-order ($m - n$) bits of a logical address designate the page number, and the n designate the page offset. Thus, the logical address is as follows:

Page Number	Page offset
p	d
m – n	n

When a process arrives for execution, its size (in terms of pages), is examined. If the process requires n pages, at least n frames must be available in memory. If n frames are available, they are allocated to this arriving process. The first page of the process is loaded into one of the allocated frames, and the frame number is put in the page table for this process. The next page is loaded into another frame, its frame number is put into the page table, and so on.

Example:



Segmentation

- *Segmentation* is a memory-management scheme that supports user view of memory.
- A logical address space is a collection of segments.
  - Each segment has a name (number) and a length (limit).
  - The addresses specify both the *segment number and the offset* within the segment.
  - Thus, a logical address format is: <*segment-number, offset*>. OR <s, d>
- Normally, Compiler automatically constructs segments for the input program.

Example: *C compiler* might create separate segments for the following:

1. The code
2. Global variables
3. The heap, from which memory is allocated
4. The stacks used by each thread (light weight process)
5. The standard C library

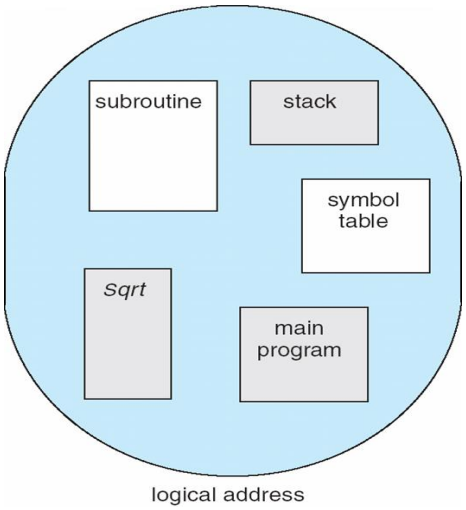


Figure: User's view of a program.

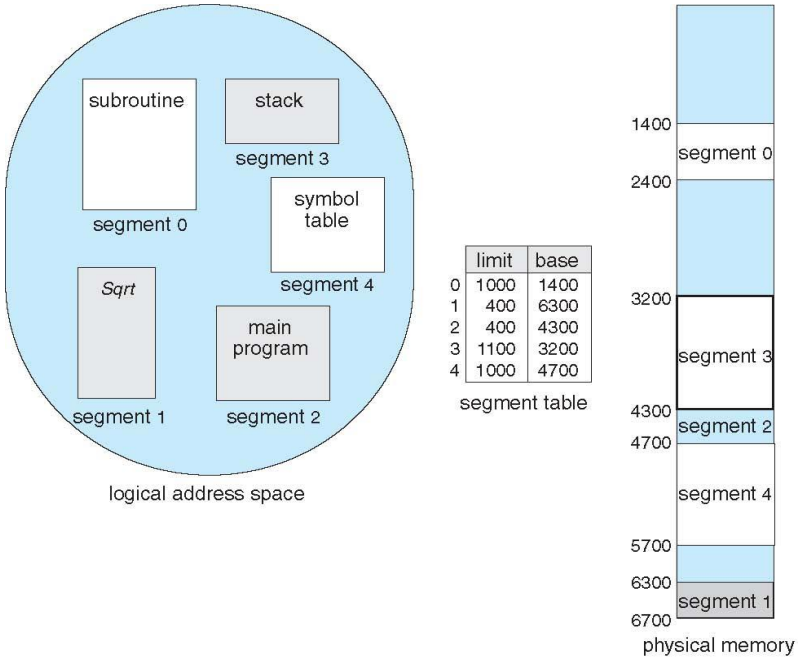


Figure: Example of segmentation.

Virtual memory

Virtual memory is a technique that allows the execution of processes that are not completely in memory.

- Virtual memory abstracts main memory into an extremely large, uniform array of storage, separating user logical memory from physical memory.
- It enables the programmers to write programs that would occupy extremely large virtual address space. Gives the impression to the users that they have large main memory.
- Only part of the program needs to be in memory for execution
- Logical address space can therefore be much larger than physical address space
- Allows address spaces / files to be shared by several processes
- Allows for more efficient process creation (using fork( ))
- Makes more programs running concurrently
- Leads to increase in CPU utilization and throughput
- Less I/O needed to load or swap processes, so each user program would run faster.
- Virtual memory offers a mechanism to accommodate one or more processes into physical memory whose address space is larger than physical memory. This is achieved by allowing partial loading of processes in main memory during execution of those processes.
  - Physical memory is divided into fixed-sized blocks called *Frames*. (size is power of 2, between 512 bytes to 16M bytes)
  - Divided the virtual (logical) memory in to fixed-sized blocks of same size (frame size) called *Pages*.

- Backing Store is divided in to fixed-sized blocks (same size as the frame size).]
- i.e. Page Size = Frame Size

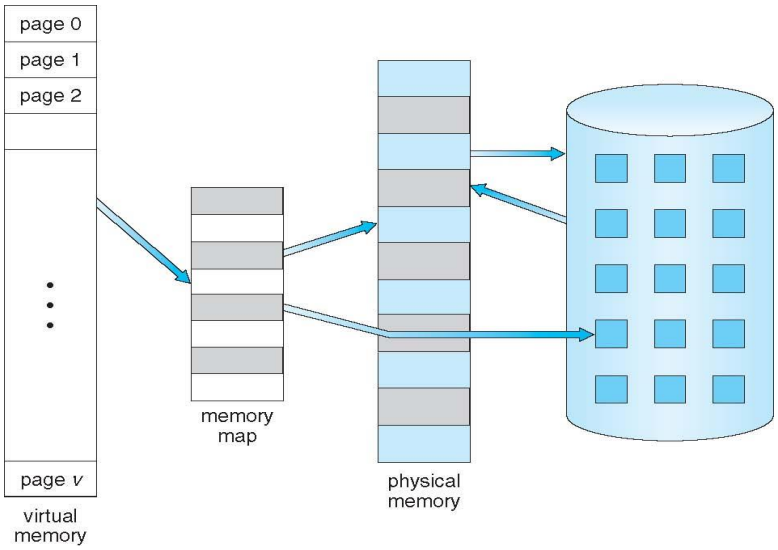


Figure: Diagram showing virtual memory that is larger than physical memory

Demand paging

A demand paging system is quite similar to a paging system with swapping. When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory, however, we use a ***lazy swapper*** called ***pager.***

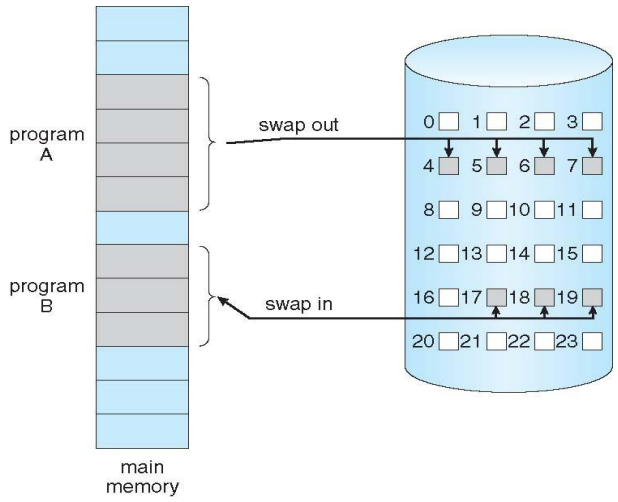


Figure: Transfer of a paged memory to Contiguous disk space.

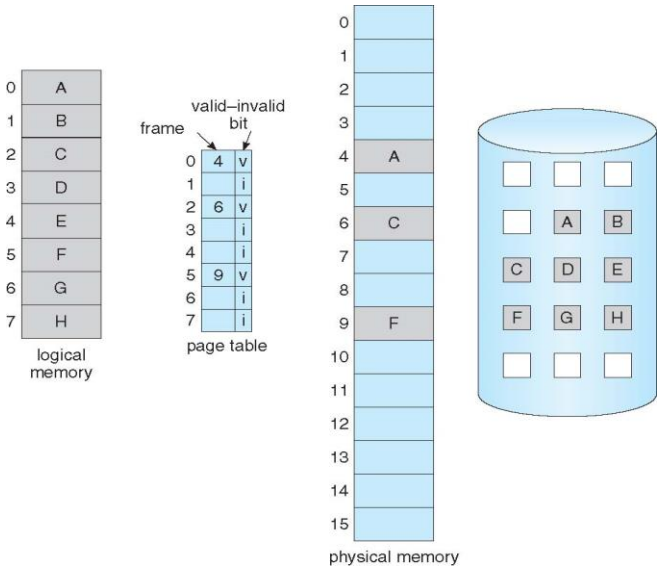


Figure: Page table when some pages are Not in main memory.

When a process is to be swapped **in**, the pager guesses which pages will be used before the process is swapped **out** again. Instead of swapping in a whole process, the pager brings only those necessary pages into memory. Thus, it avoids reading into memory pages that will not be used in anyway, decreasing the swap time and the amount of physical memory needed.

Hardware support is required to distinguish between those pages that are in memory and those pages that are on the disk using the ***valid-invalid bit scheme***. Where valid and invalid pages can be checked by checking the bit.

While the process executes and accesses pages that are in memory, execution proceeds normally. Access to a page marked invalid causes a **page-fault trap**. If the requested page is actually part of the process, then that page will be loaded from disk to main memory. If the request page is not a part of the process, then the process is terminated.

Page replacement algorithm (concept only)

1. Find the location of the desired page on disk
2. Find a free frame:
   - If there is a free frame, use it
   - If there is no free frame, use a page replacement algorithm to select a **victim frame**
     - Write victim frame to disk if dirty (i.e. modified page must be written back to disk)
3. Bring the desired page into the (newly) free frame; update the page and frame tables
4. Continue the process by restarting the instruction that caused the trap

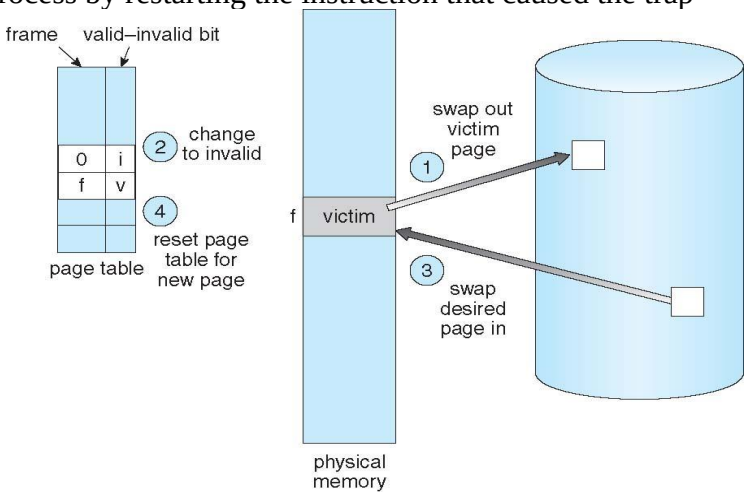
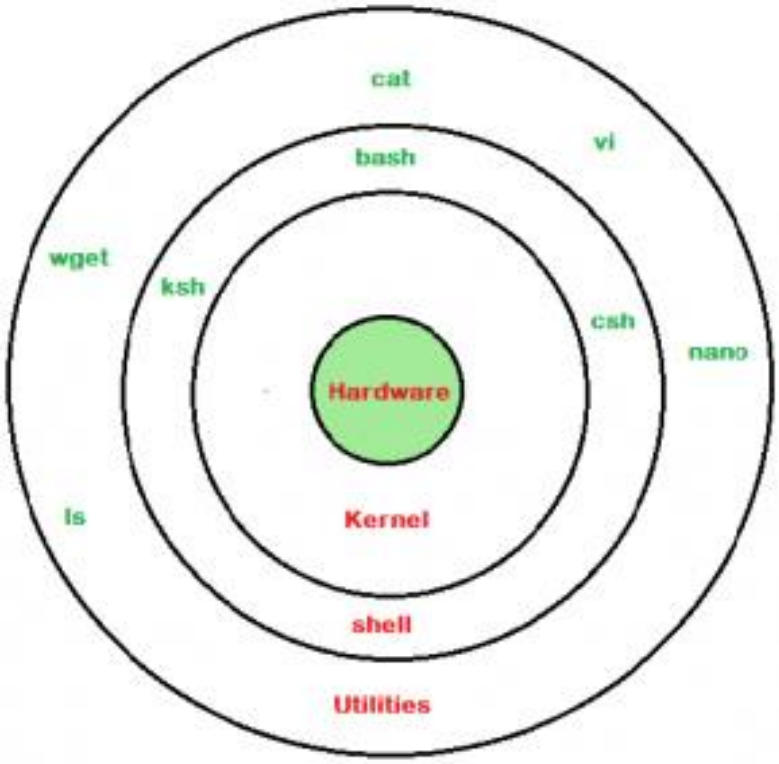


Figure: Page Replacement

Chapter 7 Shell Programming

Shell An interface to kernel, hiding complexity of kernel's functions from users. The shell takes commands from the user and executes kernel's functions.



Shell is broadly classified into two categories –

- Command Line Shell
- Graphical shell

Command Line Shell

Shell can be accessed by user using a command line interface. A special program called **Terminal** in linux/macOS or **Command Prompt** in Windows OS is provided to type in the human readable commands such as "cat", "ls" etc. and then it is being execute. The result is then displayed on the terminal to the user.

Graphical Shells

Graphical shells provide means for manipulating programs based on graphical user interface (GUI), by allowing for operations such as opening, closing, moving and resizing windows, as well as switching focus between windows. Window OS or Ubuntu OS can be considered as good example which provide GUI to user for interacting with program. User do not need to type in command for every actions.

There are several shells are available for Linux systems like:

BASH (Bourne Again SHell) – It is most widely used shell in Linux systems. It is used as default login shell in Linux systems and in macOS. It can also be installed on Windows OS.

CSH (C SHell) – The C shell's syntax and usage are very similar to the C programming language.

KSH (Korn SHell) – The Korn Shell also was the base for the POSIX Shell standard specifications etc.

Each shell does the same job but understand different commands and provide different built in functions.

Shell Scripting

Usually shells are interactive that mean, they accept command as input from users and execute

them. If there is a bunch of commands to be executed, all those commands must be typed each time in terminal.

These commands can be written in a file and can execute them in shell to avoid this repetitive work. These files are called **Shell Scripts** or **Shell Programs**. Shell scripts are similar to the batch file in MS-DOS. Each shell script is saved with **.sh** file extension eg. **myscript.sh**

A shell script have syntax just like any other programming language.

Need for shell scripts

- To avoid repetitive work and automation
- System admins use shell scripting for routine backups
- System monitoring
- Adding new functionality to the shell etc.

Advantages of shell scripts

- The command and syntax are exactly the same as those directly entered in command line, so programmer do not need to switch to entirely different syntax
- Writing shell scripts are much quicker
- Quick start
- Interactive debugging etc.

Disadvantages of shell scripts

- Prone to errors
- Slow execution
- Design flaws within the language syntax or implementation
- Not suited for large and complex task
- Provide minimal data structure unlike other scripting languages. Etc

Shebang

Shell scripts that start with **#!/bin/bash.** This **#!** is called **shebang** or **hashbang.** The shebang plays an important role in shell scripting, while dealing with different types of shell. This combination when used in the very first line of the script, specifies the interpreter with which the given script will be run by default.

If the first line of a script is: **#!/bin/bash,** It means the interpreter should be bash shell.

If the first line is: **#!/bin/zsh**, it means the interpreter to be used is Z shell.

If nothing is specified in the beginning of the script, login shell will be used for executing.

Getting Input from User

1. Get Single input

Mention variable name after read command to read and store user input in that variable.

E.g.: read name

2. Get multiple input:

E.g.: read variable1 variable2 variable3

3. Hide user input

Sometimes user may need to hide user input as they are typing. For this purpose, use read command with **-sp** option. In this case, -s stands for silent mode and -p stands for command prompt.

E.g.: read -sp password

4. Get user input without variable

If user don't use variable in read statement, then shell script will store user input in REPLY system variable.

E.g.:

```
echo "enter name"
read
echo "you entered $REPLY"
```

Displaying Output

The **echo** command is used for outputting data to a user.

E.g.: echo 'Hello world!'

Use **echo** with the option -e to use escape sequences.

\a alert
\b backspace
\c suppress new line
\n new line
\r carriage return
\t horizontal tab
\ backslash

Printing with printf

E.g.: printf "Hello world!\n"

Format specifiers

%s	strings	printf "Hello my name is %s %s.\n" Tony Dylan
%d	integers	printf "I am %d years old" 14
%f	floating point values	printf "The child is %f years old.\n" 3.5

Formatting

To limit decimal places

```
printf "%.2f\n" 3
3.00
```

To place spaces before a numerical value, do so with just an integer value.

```
printf "I am %4d years old.\n" 13
I am   13 years old.
```

To pad a number with zeros, precede with a zero.

```
printf "I am %04d years old.\n" 13
I am 0013 years old.
```

Decision making

The programmer provides one or more conditions for the execution of a block of code. If the conditions are satisfied then those block of codes only gets executed. Two types of decision-making statements are used within shell scripting. They are –

- If-else statement
- case-sac statement

1. If-else statement

If else statement is a conditional statement. It can be used to execute two different codes based on whether the given condition is satisfied or not. There are varieties ofthe if-else statement. They are –

- if-fi
- if-else-fi
- if-elif-else-fi
- nested if-else

The syntax will be –

if-fi

```
if [ expression ]; then
   statements
fi
```

if-else-fi

```
if [ expression ]
then
   statement1
else
   statement2
fi
```

if-elif-else-fi

```
if [ expression1 ]
then
   statement1
   statement2
   .
   .
elif [ expression2 ]
then
   statement3
   statement4
   .
   .
else
   statement5
fi
```

nested if-else

```
if [ expression ]
then
   statement1
   if [ expression ]
   then
     statement
   else
     statement
   fi
else
   statement2
fi
```

2. The case-esac statement

case-esac is basically working the same as switch statement in programming. Sometimes if user has to check multiple conditions,use a case-esac statement. The syntax will be –

```
case $var in
   Pattern 1) Statement 1;;
   Pattern n) Statement n;;
esac
```

E.g.:

```
Name="Satyajit"
case "$Name" in
    #case 1
    "Rajib") echo "Profession : Software Engineer" ;;
    #case 2
    "Vikas") echo "Profession : Web Developer" ;;
     #case 3
    "Satyajit") echo "Profession : Technical Content Writer" ;;
```

esac

Output

Profession : Technical Content Writer

Iterative Scripts (Looping)

There are total 3 looping statements which can be used in bash programming

1.while statement

2. for statement

3. until statement

To alter the flow of loop statements, two commands are used they are,

1.break

2. continue

Their descriptions and syntax are as follows:

while statement

Here command is evaluated and based on the result loop will executed, if command raise to false then loop will be terminated

Syntax

while condition
do
  Statement(s) to be executed if command is true
done

for statement

The for loop operate on lists of items. It repeats a set of commands for every item in a list. Here var is the name of a variable and word1 to wordN are sequences of characters separated by spaces (words). Each time the for loop executes, the value of the variable var is set to the next word in the list of words, word1 to wordN

Syntax

for var in word1 word2 ... wordN
do
  Statement(s) to be executed for every word.
done

until statement

The until loop is executed as many as times the condition/command evaluates to false. The loop terminates when the condition/command becomes true.

Syntax

until command
do
  Statement(s) to be executed until command is true
done

Chapter 8 Automation of system tasks

Sometimes, user may have tasks that need to be performed on a regular basis or at certain predefined intervals. Such tasks include backing up databases, updating the system, performing periodic reboots daily backups, monthly log archiving, and weekly file deletion to create space and so on. Such tasks in linux are referred to as **cron jobs (Crontab).** Cron jobs are used for **automation of tasks** that come in handy and help in simplifying the execution of repetitive and sometimes everyday tasks.

Commands to Schedule Tasks:

cron:

- The **cron** is a software utility, offered by a Linux-like operating system that automates the scheduled task at a predetermined time.
- It is a **daemon process**, which runs as a background process and performs the specified operations at the predefined time when a certain event or condition is triggered without the intervention of a user.
- Dealing with a repeated task frequently is an intimidating task for the system administrator and thus he can schedule such processes to run automatically in the background at regular intervals of time by creating a list of those commands using cron.
- It enables the users to execute the scheduled task on a regular basis unobtrusively like doing the backup every day at midnight, scheduling updates on a weekly basis, synchronizing the files at some regular interval.
- Cron checks for the scheduled job recurrently and when the scheduled time fields match the current time fields, the scheduled commands are executed.
- It is started automatically from /etc/init.d on entering multi-user run levels.
- **Syntax: cron [-f] [-l] [-L loglevel]**
- The **crontab** (abbreviation for "cron table") is list of commands to execute the scheduled tasks at specific time. It allows the user to add, remove or modify the scheduled tasks.
- The crontab command syntax has **six fields** separated by space where the **first five** represent the **time to run the task** and **the last one is for the command**.
    - Minute (holds a value between 0-59)
    - Hour (holds value between 0-23)
    - Day of Month (holds value between 1-31)
    - Month of the year (holds a value between 1-12 or Jan-Dec, the first three letters of the month's name shall be used)
    - Day of the week (holds a value between 0-6 or Sun-Sat, here also first three letters of the day shall be used)
    - Command (**6th Field**)

The rules which govern the format of date and time field as follows:

- When any of the first five fields are set to an asterisk(*), it stands for all the values of the field. For instance, to execute a command daily, we can put an asterisk(*) in the week's field.
- One can also use a range of numbers, separated with a hyphen(-) in the time and date field to include more than one contiguous value but not all the values of the field. For example, we can use the 7-10 to run a command from July to October.
- The comma (,) operator is used to include a list of numbers which may or may not be consecutive. For example, "1, 3, 5" in the weeks' field signifies execution of a command every Monday, Wednesday, and Friday.
- A slash character(/) is included to skip given number of values. For instance, "*/4" in the hour's field specifies 'every 4 hours' which is equivalent to 0, 4, 8, 12, 16, 20.

Permitting users to run cron jobs:

- The user must be listed in this file to be able to run cron jobs if the file exists.
    - **/etc/cron.allow**
- If the cron.allow file doesn't exist but the cron.deny file exists, then a user must not be listed in this file to be able to run the cron job.
    - **/etc/cron.deny**
- **Note:** If neither of these files exists then only the superuser(system administrator) will be allowed to use a given command.

Example to Try:

- As cron processes will run in background, so to check whether scripts are executed or not, one way to check the log file i.e. /var/log/syslog
- Search for cron in syslog using command: **cat /var/log/syslog/ | grep cron**
- User can see all the entries, which shows, the scripted executed at a scheduled time mentioned in crontab file.

Simple Example 1:

File Name: task.sh path: **/home/admincs/**(say for example)

Contents:

```
echo  Welcome to Task Scheduler Demo
echo  Try date Command
date
echo  Try time Command
time
echo  List all file/folders in a home directory
ls
echo  End of Task Scheduler Demo
```

=== * ===

- Change the execution permission to task.sh

  chmod 764 task.sh

- Add the task to /etc/crontab (Task will be executed at 4.13pm, User can Change timings according to the requirements)

  13 16 * * * root sh /home/admincs/task.sh > /home/admincs/output.txt

- Save and Exit. And Wait till 4.13pm
- After that Check the output.txt by the command

  cat /home/admincs/output.txt

- Output of the task.sh is present in output.txt.
- Alternatively **/var/log/syslog** can be verified about the execution of the Task.

Chapter 9 Network Management

Network components

IP (Internet Protocol) address is as a unique identifier for each device on the Internet. Length of the IP address is 32-bits. IPv6 address is 128 bits.

These 32 bits are divided into four octets, of eight bits each. Each of these four octets is represented in a decimal form, and separated by a dot.

E.g.: 198.172.168.10

This format of representing IP address is called the dotted decimal format. The octets in an IP address can take a decimal value from 9 to 255 because the largest decimal value that can be represented by eight binary bits is 255(11111111 in binary).

E.g.: The 32 bit binary address 11000110.10101100.1010100.0001010 represents the IP address 198.172.168.10

IPv6 is the next generation of IP addresses. The main difference between IPv4 and IPv6 is the address size of IP addresses. The IPv4 is a 32-bit address, whereas IPv6 is a 128-bit hexadecimal address. IPv6 provides a large address space, and it contains a simple header as compared to IPv4.

It provides transition strategies that convert IPv4 into IPv6, and these strategies are as follows:

- **Dual stacking:** It allows us to have both the versions, i.e., IPv4 and IPv6, on the same device.
- **Tunneling:** In this approach, all the users have IPv6 communicates with an IPv4 network to reach IPv6.
- **Network Address Translation:** The translation allows the communication between the hosts having a different version of IP.

This hexadecimal address contains both numbers and alphabets. Due to the usage of both the numbers and alphabets, IPv6 is capable of producing over 340 undecillion (3.4*1038) addresses. IPv6 is a 128-bit hexadecimal address made up of 8 sets of 16 bits each, and these 8 sets are separated by a colon. In IPv6, each hexadecimal character represents 4 bits. So, we need to convert 4 bits to a hexadecimal number at a time

IP Subnetting

Sub netting is the process of breaking down an IP network into smaller sub-networks called ―subnets. Each subnet is a non-physical description (or ID) for a physical sub-network.

Subnet Mask

Every device has an IP address with two pieces: the **client or host address** and the **server or network address**. IP addresses are either configured by a DHCP server or manually configured (static IP addresses).

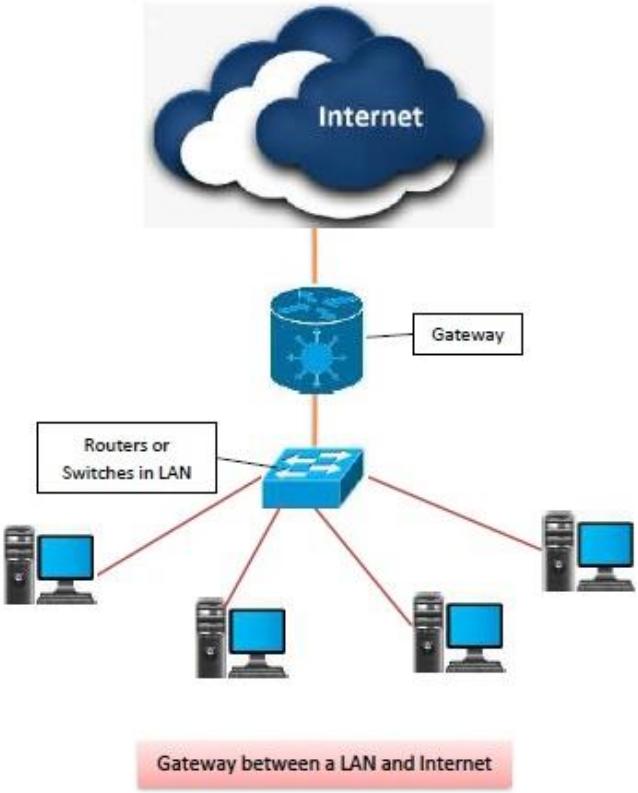
The subnet mask splits the IP address into the host and network addresses, thereby defining which part of the IP address belongs to the device and which part belongs to the network.

The device called a gateway or default gateway connects local devices to other networks. This means that when a local device wants to send information to a device at an IP address on another network, it

first sends its packets to the gateway, which then forwards the data on to its destination outside of the local network.

Gateway

A gateway is a network node that forms a passage between two networks operating with different transmission protocols. The network gateway operates at layer 3, i.e. network layer of the OSI (open systems interconnection) model. However, depending upon the functionality, a gateway can operate at any of the seven layers of OSI model. It acts as the entry – exit point for a network since all traffic that flows across the networks should pass through the gateway. Only the internal traffic between the nodes of a LAN does not pass through the gateway.



Gateway between a LAN and Internet

Types of Gateways

On basis of direction of data flow, gateways are divided into two categories −

Unidirectional Gateways − allow data to flow in only one direction.

Bidirectional Gateways − they allow data to flow in both directions.

Network Interface management

The first step in network management is collecting information from the network that is being managed. The information comes from network devices, such as routers, switches and wireless access points. Data can also be collected via software-based agents in a virtual or cloud service that provide visibility into network and application data traffic flows.

Protocols used for network management:

SNMP. Simple Network Management Protocol (SNMP) is commonly deployed on networking devices as a decades-old approach for obtaining network information.

NETCONF. Network Configuration Protocol (NETCONF) is a protocol for network configuration that provides mechanisms for network management tools and administrators to configure connected network devices.

RESTCONF. The RESTCONF protocol builds on top of NETCONF using a RESTful application programming interface-based approach to update and change network configuration.

gNMI. The gRPC Network Management Interface (gNMI) protocol provides a way to obtain the given state for a network device or service, as well as the ability to manipulate and modify the configuration for the device.

Data transfer facilitation

Chapter 10 User authentication

Work on user accounts:

useradd

- **useradd** is a command in Linux that is used to add user accounts to system.
- It is just a symbolic link to **adduser** command in Linux and the difference between both of them is that useradd is a native binary compiled with system whereas adduser is a Perl script which uses useradd binary in the background.
- When we run the '**useradd**' command in the Linux terminal, it performs the following major things:
    - It edits **/etc/passwd**, **/etc/shadow**, **/etc/group** and **/etc/gshadow** files for the newly created user accounts.
    - Creates and populates a home directory for the new user.
    - Sets permissions and ownerships to the home directory.
- **Syntax:**
    - **useradd -d /home/newusername –p passwordstring newusername**
    - This creates user with name newusername
        - -d option will create directory for new user at /home directory
        - -p option allows specifying password while creating user with the value given in place of passwordstring
    - User can change the password later by typing command
        - passwd newusername
        - This asks the user to enter the new password and confirm password

=== * ===

passwd

- The passwd command **changes passwords for user accounts**.
- A normal user may only change the password for their own account, while the superuser may change the password for any account.
- passwd also changes the account or associated password validity period.
- **Important options** used along with passwd:
    - **-d, --delete** Delete a user's password (make it empty). This is a quick way to disable a password for an account. It will set the named account passwordless.
    - **-e, --expire** Immediately expire an account's password. This in effect can force a user to change his/her password at the user's next login.

=== * ===

userdel

- **userdel** command in Linux system is used to delete a user account and related files.
- This command basically modifies the system account files, deleting all the entries which refer to the username LOGIN.
- It is a low-level utility for removing the users.
- **Syntax: sudo userdel -f –r –Z neuser**
- **Important Options** used along with userdel:
    - **-f:** This option forces the removal of the specified user account. It doesn't matter that the user is still logged in. It also forces the *userdel* to remove the user's home directory and mail spool, even if another user is using the same home directory or even if the mail spool is not owned by the specified user.
    - **-r:** Whenever we are deleting a user using this option then the files in the user's home directory will be removed along with the home directory itself and the user's mail spool. All the files located in other file systems will have to be searched for and deleted manually.
    - **-Z :** This option remove any SELinux(Security-Enhanced Linux) user mapping for the user's login.

=== * ===

usermod

- usermod command or modify user is used to change the properties of a user.
- After creating a user we have to sometimes change their attributes like password or login directory etc.
- To change the home directory of a user
    - **usermod -d /home/newfolder existinguser**
- To change the group of a user
    - **usermod -g newgroupname existinguser**
- To change user login name
    - **usermod -l usernewname useroldname**
- To lock a user
    - **usermod -L existinguser** // -U for unlocking user
- To set an unencrypted password for the user
    - **usermod -p newpassword existinguser**

=== * ===

Groups: Groups in Linux refer to the user groups. In Linux, there can be many users of a single system, (normal user can take uid from 1000 to 60000, and one root user (uid 0) and 999 system users (uid 1 to 999)).

groupadd

- **groupadd** command is used to create a new user group.
- **Syntax**: **groupadd newgroupname**
- Every new group created is registered in the file "/etc/group". To verify that the group has been created, enter the command
    - **sudo tail /etc/group**
- Adding user while creating newuser
    - **useradd -d /home/newusername –p passwordstring -g existinggroupname newusername**
- Adding existing user to group // Explained once again in next topic
    - **usermod -g existinggroupname existinguser**

=== * ===

groupmod

- **groupmod** command is used to modify or change the existing group.
- It can be handled by superuser or root user.
- Basically, it modifies a group definition on the system by modifying the right entry in the database of the group.
- **Syntax: groupmod [option] groupname**
    - Change the group *name to new name*.
    - **groupmod -n groupnewname groupoldname**

=== * ===

gpasswd

- *gpasswd* command is used to administer the */etc/group* and */etc/gshadow*.
- *gpasswd* command **assigns** a user to a group with some security criteria.
- *gpasswd* command is called by a group administrator with a group name only which prompts for the new password of the group.
- System administrators can use the *-A* option to define group administrator(s) and *-M* option to define members.
- **Syntax: gpasswd [option] group**
- **Important Options:** Here only -A and -M options can be combined.
    - **-a, –add :** This option is used to add a user to the named group.

- **-d, –delete :** It is used to remove a user from the named group.
- **-r, –remove-password :** It is used to remove the password from the named group.
- **-A, –administrators :** Set the list of administrators for the group.
- **-M, –members : set the list of members of the group**.

Example:

- add the user to group:
  - **gpasswd -a existinguser existinggroup**
- Deleting the created user from group geeks.
  - **gpasswd -d existinguser existinggroup**

=== * ===

groupdel

- *groupdel* command is used to delete a existing group.
- It will delete all entry that refers to the group, modifies the system account files, and it is handled by superuser or root user.
- **Syntax: groupdel –f existinggroup**
- **Option used: -f –force**: It used to delete a group even if it is the primary group of a user

Linux Directory Service

Directory services are a standard feature of any medium to large corporate network. The role of a directory service is to make administering and navigating a large network much more manageable. Network-wide functions such as authentication, user databases and centralised file repositories can all be provided using a directory service.

LDAP is a tool for providing directory services for a network on a Linux server.

LDAP server and client configuration:

- Lightweight Directory Access Protocol (LDAP) is a standard protocol designed to manage and access hierarchical directory information over a network.
- It can be used to store any kind of information, though it is most often used as a centralized authentication system or for corporate email and phone directories.
- Use phpLDAPadmin, a web interface for viewing and manipulating LDAP information.

LDAP structure

An LDAP directory has a tree structure. All entries (called objects) of the directory have a defined position within this hierarchy. This hierarchy is called the directory information tree or, for short, DIT. The complete path to the desired entry, which unambiguously identifies it, is called distinguished name or DN. The single nodes along the path to this entry are called relative distinguished name or RDN. Objects can generally be assigned to one of two possible types:

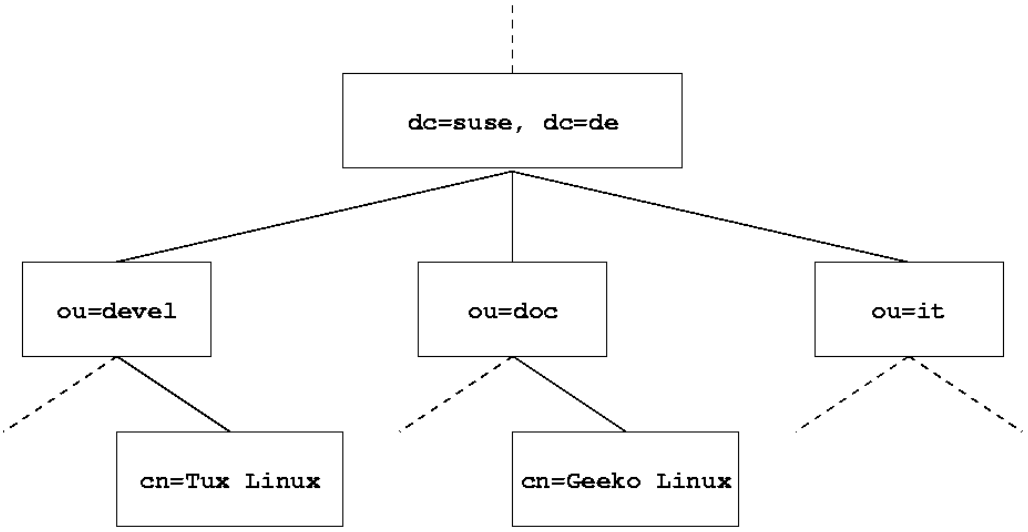
Container

These objects can themselves contain other objects. Such object classes are root (the root element of the directory tree, which does not really exist), c (country), ou (organizational unit), and dc (domain component). This model is comparable to the directories (folders) in a file system.

Leaf

These objects sit at the end of a branch and have no subordinate objects. Examples are person, InetOrgPerson, or groupofNames.

The top of the directory hierarchy has a root element root. This can contain c (country), dc (domain component), or o (organization) as subordinate elements. The relations within an LDAP directory tree become more evident in the following example, shown in Figure



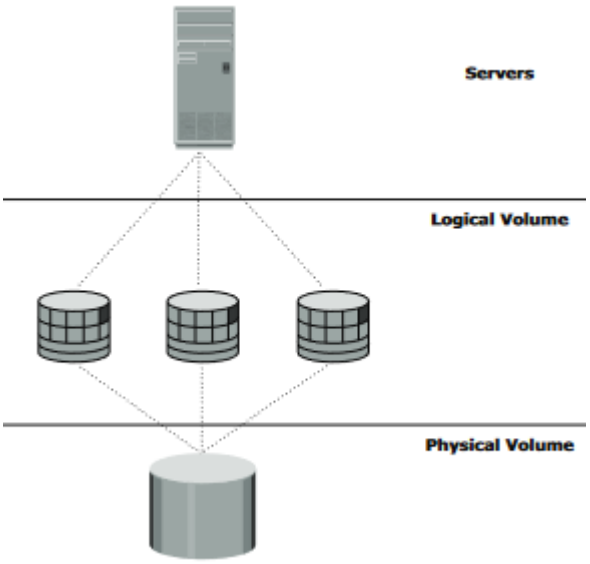
Commonly Used Object Classes and Attributes

Object Class	Meaning	Example Entry	Compulsory Attributes
dcObject	*domainComponent* (name components of the domain)	suse	dc
organizationalUnit	*organizationalUnit* (organizational unit)	doc	ou
inetOrgPerson	*inetOrgPerson* (person-related data for the intranet or Internet)	Geeko Linux	sn and cn

Chapter 13 Storage management

Disk partition

Disk partitioning was introduced to improve the flexibility and utilization of HDDs. In partitioning, an HDD is divided into logical containers called logical volumes (LVs).



A large physical drive can be partitioned into multiple LVs to maintain data according to the file system's and applications' requirements. The partitions are created from groups of contiguous cylinders when the hard disk is initially set up on the host. The host's file system accesses the partitions without any knowledge of partitioning and the physical structure of the disk.

Logical Volume Management (LVM)

LVM is software that runs on the host computer and manages the logical and physical storage. It can aggregate several smaller disks to form a larger virtual disk or to partition a larger-capacity disk into virtual, smaller-capacity disks, which are then presented to applications. It hides details about the physical disk and the location of data on the disk; and it enables administrators to change the storage allocation without changing the hardware, even when the application is running.

LVM components

- Physical volumes
- Volume groups
- Logical volumes.

Each physical disk connected to the host system is a physical volume (PV). LVM converts the physical storage provided by the physical volumes to a logical view of storage, which is then used by the operating system and applications.

A **volume group** is created by grouping together one or more physical volumes. A unique physical volume identifier (PVID) is assigned to each physical volume when it is initialized for use by the LVM.

Physical volumes can be added or removed from a volume group dynamically. They cannot be shared between volume groups; the entire physical volume becomes part of a volume group.

Each physical volume is partitioned into equal-sized data blocks called **physical extents** when the volume group is created.

Logical volumes are created within a given volume group. A **logical volume** can be thought of as a virtual disk partition, while the volume group itself can be thought of as a disk. A volume group can have a number of logical volumes. The size of a logical volume is based on a multiple of the physical extents. The logical volume appears as a physical device to the operating system. A logical volume can be made

up of non-contiguous physical partitions and can span multiple physical volumes. A file system can be created on a logical volume; and logical volumes can be configured for optimal performance to the application and can be mirrored to provide enhanced data availability.

Extend Disk using LVM

Use **lvextend** command to increase the size.

RAID (Redundant Arrays of Inexpensive/ independent Disks

RAID is an enabling technology that leverages multiple disks as part of a set which provides data protection against HDD failures. RAID implementations also improve the I/O performance of storage systems by storing data across multiple HDDs.

Implementation of RAID

There are two types of RAID implementation, hardware and software.

Software RAID

Software RAID uses host-based software to provide RAID functions. It is implemented at the operating-system level and does not use a dedicated hardware controller to manage the RAID array. Software RAID implementations offer cost and simplicity benefits when compared with hardware RAID.

Limitations

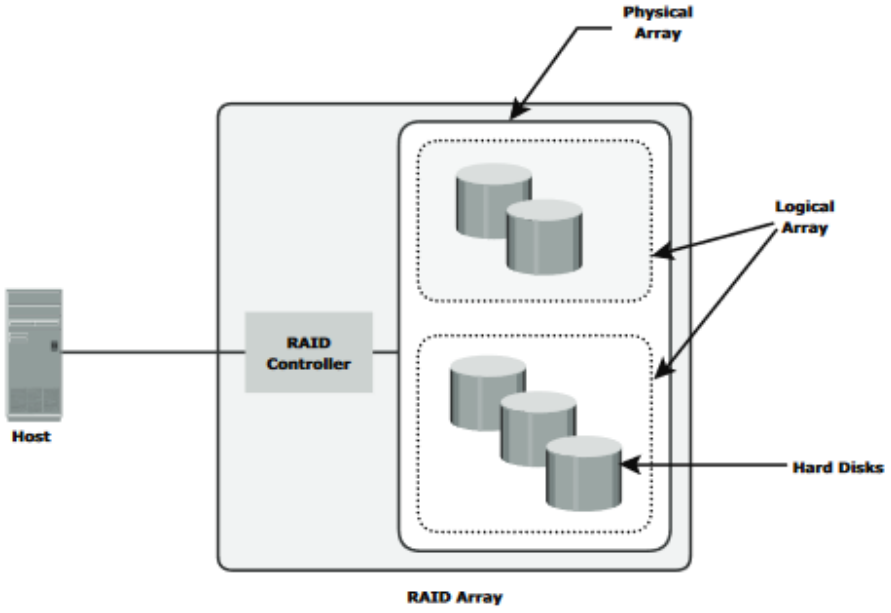
Performance: Software RAID affects overall system performance. This is due to the additional CPU cycles required to perform RAID calculations.

Supported features: Software RAID does not support all RAID levels.

Operating system compatibility: Software RAID is tied to the host operating system hence upgrades to software RAID or to the operating system should be validated for compatibility. This leads to inflexibility in the data processing environment.

Hardware RAID

In hardware RAID implementations, a specialized hardware **controller** is implemented either on the host or on the array. These implementations vary in the way the storage array interacts with the host. Controller card RAID is host-based hardware RAID implementation in which a specialized RAID controller is installed in the host and HDDs are connected to it. The RAID Controller interacts with the hard disks using a PCI bus. Manufacturers also integrate RAID controllers on motherboards. This integration reduces the overall cost of the system, but does not provide the flexibility required for high-end storage systems.



RAID Levels

LEVELS	BRIEF DESCRIPTION
RAID 0	Striped array with no fault tolerance
RAID 1	Disk mirroring
RAID 3	Parallel access array with dedicated parity disk
RAID 4	Striped array with independent disks and a dedicated parity disk
RAID 5	Striped array with independent disks and distributed parity
RAID 6	Striped array with independent disks and dual distributed parity
Nested	Combinations of RAID levels. Example: RAID 1 + RAID 0