1.1 History of Operating Systems

HNDIT3052
Operating Systems

History of Operating Systems

- **Generations:**
  - First Generation (1945 - 1955): Vacuum Tubes and Plugboards.
  - Second Generation (1955 - 1965): Transistors and Batch Systems.
  - Third Generation (1965 - 1980): Integrated Circuits and Multiprogramming.
  - Fourth Generation (1980 - Present): Personal Computers.
  - The Fifth Generation (1990–Present): Mobile Computers, Tablets, Smart Phones.

First Generation (1945 - 1955): Vacuum Tubes and Plugboards

- 1945-1955, 2nd world war, no digital computers,.
- Calculating engines which used mechanical relays were built.
- These mechanical relays were extremely slow.
- later on replaced by comparatively faster vacuum tubes.
- designed and built by a single group of people.
- At that time,
  - programming languages were unknown.
  - there was no such thing as an operating system. (because of which all the programming was done in machine languages).
- Problems solved were simple numerical calculations.
- In the 1950s:
  - Punch cards were introduced , and they improved the overall computer system.
  - Now instead of using plugboards, **programs were written** on cards and were handed to the operator, who fed them to the system.

Second Generation (1955 - 1965): Transistors and Batch Systems

- took place around 1955 to 1965.
- **GMO:** The first operating system, GMOs was created in the early 1950s. General Motors developed this OS for the IBM computers.

  Mainframes: In this period, transistors were developed, which led to the development of computer systems that could be manufactured and sold to paying customers. Such types of machines were known as mainframes. They were kept in air-conditioned rooms with trained staff to operate them.
- **batch operating system:**
  - Later, Batch systems were introduced, in this to reduce the computer's idle time.
  - All the jobs were collected on a tray in the input room and were read into the magnetic tape. Then, the tape was mounted onto a tape drive once it had been rewound. When the batch operating system was loaded, it read the first job from the tape and ran it, while the output was written onto the second tape. After executing the whole batch, the input and output tapes were removed while the output tape was printed.

Transistors and Batch Systems (1)

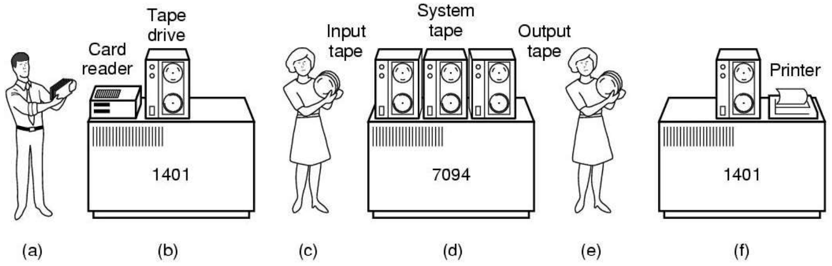


Figure 1-3. An early batch system.
(a) Programmers bring cards to 1401.
(b)1401 reads batch of jobs onto tape.

Transistors and Batch Systems (2)

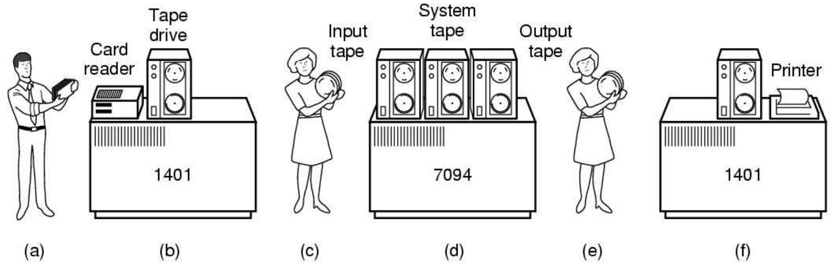


Figure 1-3. (c) Operator carries input tape to 7094.
(d) 7094 does computing. (e) Operator carries output tape to 1401. (f) 1401 prints output.

Transistors and Batch Systems (4)

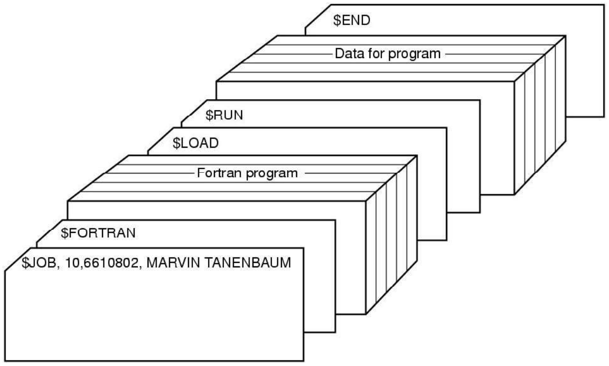


Figure 1-4. Structure of a typical FMS job.

Third Generation (1965 - 1980): Integrated Circuits and Multiprogramming.

- 1965-1980 **(System/360)**.
  - Initially, there were two types of computers (served a different purpose).
  - The scientific and commercial computers.
  - These were combined into one by IBM, which was known as the **System/360**.
  - They used integrated circuits. (offered a major price and performance advantage over the second-generation systems).
- By the late 1960s (multiprogramming operating systems).
  - OS designers were capable of developing an operating system that was capable of performing multiple tasks simultaneously in a single computer.
  - Such operating systems were known as multiprogramming operating systems.
  - The introduction of multiprogramming played an important role in developing operating systems that allowed the CPU to be busy all the time by performing different tasks on a single computer simultaneously.

ICs and Multiprogramming

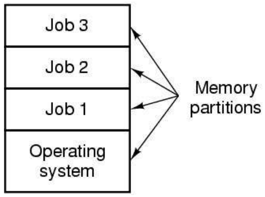


Figure 1-5. A multiprogramming system with three jobs in memory.

Fourth Generation (1980 - Present): Personal Computers.

- Personal Computers (PCs):
  - around 1980 and is being used till now:
  - Operating systems is related to the development of personal computers.
  - Computers similar to the minicomputers in 3rd generation, the cost of PCs was very high.

Windows Operating System:

- The birth of **Microsoft** and windows operating systems.
- Played a major role in the development of personal computers.
- Microsoft created the first window operating system in 1975. (Bill Gates and Paul Allen).

DOS Operating System:

- Bill Gates and Paul Allen introduced MS-DOS in 1981.
- very difficult person to understand its commands.

Multiple Operating systems by Microsoft:

- After that, multiple operating systems were released by windows, such as Windows 95, Windows 98, Windows XP, etc.
- Currently, most Windows users use the Windows 10 operating system.

Macintosh OS by Apple:

- Apart from windows, Apple also built an operating system in the 1980s, which is known as Macintosh OS. it was developed by Steve Jobs, co-founder of Apple.

More in third generation

- Time Sharing (CTSS)
- Multics
- Unix
- Linux

Fourth generation

- PCs
- Network Operating Systems
- Distributed Operating Systems

The Fifth Generation (1990–Present): Mobile Computers

- In 1940's: talking to his ''two-way radio wrist watch'' in the 1940s comic strip. (people could carry around wherever they went).
- 1946: first real mobile phone appeared in 1946 and weighed some 40 kilos.
- In 1970s: first true handheld phone appeared in the 1970s and, at roughly one kilogram (known as ''the brick.').
- Mid 1990: first real smartphone did not appear until the mid-1990s when Nokia released the N9000 [combined version - a phone and a **PDA** (Personal Digital Assistant)].
- In 1997: Ericsson coined the term *smartphone* for its GS88 ''Penelope.''

Computers in 5th Generation

- The period of fifth generation is 1980-till date. In the fifth generation, VLSI technology became ULSI (Ultra Large Scale Integration) technology, resulting in the production of microprocessor chips having ten million electronic components.
- This generation is based on parallel processing hardware and AI (Artificial Intelligence) software. AI is an emerging branch in computer science, which interprets the means and method of making computers think like human beings. All the high-level languages like C and C++, Java, .Net etc., are used in this generation.

5th generation OS.. : Smart Phones and its Operating System's evolution.

- Now that smartphones have become ubiquitous.
- Google's Android is the dominant operating system with Apple's iOS.
- **Smart phones with Symbian** OS: For Samsung, Sony Ericsson, Motorola, and especially Nokia.
- **RIM's** Blackberry OS (introduced for smartphones in 2002).
- Apple's iOS (released for the first **iPhone** in 2007).
- In 2011, Nokia ditched **Symbian** and announced it would focus on Windows Phone, but, Linux-based operating system released by Google in 2008, to overtake.
- **Android:** Android had the advantage that it was open source and available under a permissive license.
  - their own hardware with ease.
  - has a huge community of developers for app writing (as most familiar with Java).

AI in 5th Generation

- **AI includes :–**
  - Robotics, Neural Networks, Game Playing, Development of expert systems to make decisions in real-life situations, Natural language understanding and generation.
- **The main features of fifth generation are :–**
  - ULSI technology, Development of true artificial intelligence, Development of Natural language processing, Advancement in Parallel Processing, Advancement in Superconductor technology, More user-friendly interfaces with multimedia features, Availability of very powerful and compact computers at cheaper rates.
- **Computer types of this generation are :–**
  - Desktop, Laptop, NoteBook, UltraBook, ChromeBook

Operating System Timeline:

OS Timeline (60s & early 70s)

1966
- DOS/360 (IBM)
- MS/8

1967
- ACP (IBM)
- CP/CMS
- ITS
- WAITS

1969
- TENEX
- Unix

1970s

1970
- DOS/BATCH 11 (PDP-11)

1971
- OS/8

1972
- MFT (Operating System)
- MVT
- RDOS
- SVS
- VM/CMS

OS Timeline (late 70s)

1973
- Alto OS
- RSX-11D
- RT-11
- VME

1974
- MVS (MVS/XA)

1975
- BS2000

1976
- CP/M
- TOPS-20

1978
- Apple DOS 3.1 (*first Apple OS*)
- TripOS
- VMS
- Lisp Machine (CADR)

1979
- POS
- NLTSS

OS Timeline (50s & early 60s)

1956
- GM-NAA I/O

1959
- SHARE Operating System

1960
- IBSYS

1961
- CTSS
- MCP

1962
- GCOS

1964
- EXEC 8
- OS/360 (announced)
- TOPS-10

1965
- Multics (announced)
- OS/360 (shipped)
- Tape Operating System (TOS)

OS Timeline (80s)

1980
- OS-9
- QDOS
- SOS
- XDE (Tajo) (*Xerox Development Environment*)
- Xenix

1981
- MS-DOS

1982
- Commodore DOS
- SunOS (1.0)
- Ultrix

1983
- Lisa OS
- Coherent
- Novell Netware
- ProDOS

1984
- Macintosh OS (*System 1.0*)
- MSX-DOS
- QNX
- UniCOS

OS Timeline (late 80s)

1985
- AmigaOS
- Atari TOS
- MIPS OS
- Microsoft Windows 1.0 (*First Windows*)

1986
- AIX
- GS-OS
- HP-UX

1987
- Arthur
- IRIX (*3.0 is first SGI version*)
- Minix
- OS/2 (1.0)
- Microsoft Windows 2.0

1988
- A/UX (Apple Computer)
- LynxOS
- MVS/ESA
- OS/400

1989
- NeXTSTEP (1.0)
- RISC OS
- SCO Unix (*release 3*)

OS Timeline (90s)

1990
- Amiga OS 2.0
- BeOS (v1)
- OSF/1
- Windows 3.0

1991
- Linux

1992
- 386BSD 0.1
- Amiga OS 3.0
- Solaris (*2.0 is first not called SunOS*)
- Windows 3.1

1993
- Plan 9 (First Edition)
- FreeBSD
- NetBSD
- Windows NT 3.1 (*First version of NT*)

1995
- Digital UNIX (*aka* Tru64)
- OpenBSD
- OS/390
- Windows 95

OS Timeline (late 90s and 2000)

1996
- Windows NT 4.0

1997
- Inferno
- Mac OS 7.6 (*first officially-named Mac OS*)
- SkyOS

1998
- Windows 98

1999
- AROS (Boot for the first time in Stand Alone version)
- Mac OS 8

2000
- AtheOS
- Mac OS 9
- MorphOS
- Windows 2000
- Windows Me

OS Timeline (90s)

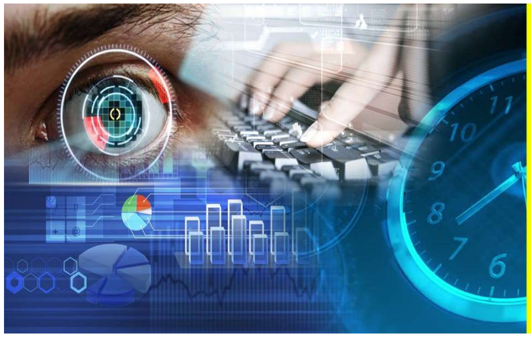
2001
- Amiga OS 4.0 (May 2001)
- Mac OS X 10.1
- Windows XP
- z/OS

2002
- Syllable
- Mac OS X 10.2

2003
- Windows Server 2003
- Mac OS X 10.3

2005
- Mac OS X 10.4

Exercise: Complete the OS Timeline for other ranges from 2006 up to now.



HNDIT 3052 Operating Systems

Types of Operating Systems

HNDIT Introduction to Operating Systems

- Week 1 Topic: Introduction to Operating System.
- Subtopics:
  - (01) History of Operating Systems.
  - (02) Types of Operating Systems.
  - (03) Operating System Concepts.

What is an Operating System (1)

- **Definition 1:** An operating system acts as an intermediary between the user of a computer and the computer hardware. The purpose of an operating system is to provide an environment in which a user can execute programs in a convenient and efficient manner.
- An operating system is software that manages the computer hardware. The hardware must provide appropriate mechanisms to ensure the correct operation of the computer system and to prevent user programs from interfering with the proper operation of the system.

What is an Operating System? (2)

- **Definition 2**: For the following reasons, computers are equipped with a layer of software called the **operating system**, whose job is to provide user programs with a better, simpler, cleaner, model of the computer and to handle managing all the resources just mentioned.
- A modern computer consists of one or more processors, some main memory, disks, printers, a keyboard, a mouse, a display, network interfaces, and various other input/output devices.
- All in all, a complex system. If every application programmer had to understand how all these things work in detail, no code would ever get written. Furthermore, managing all these components and using them optimally is an exceedingly challenging job.

02. Types of Operating Systems

2. Types of Operating Systems

- Here are the different types of operating systems you need to know:

1. **Batch OS**
2. **Time-sharing or multitasking OS**
3. **Distributed OS**
4. **Network OS**
5. **Real-time OS**
6. **Mobile OS**

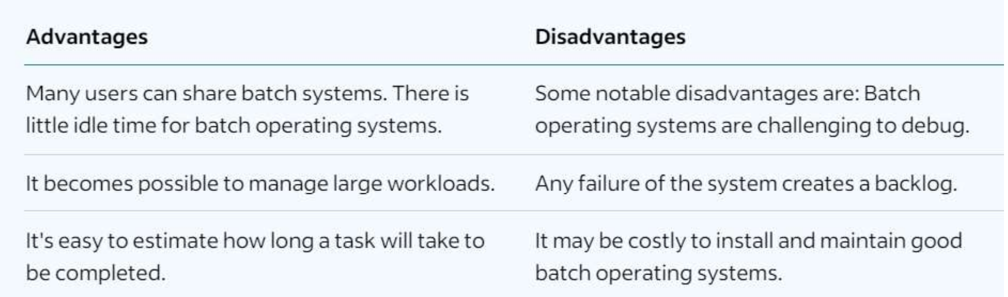
What are operating systems?

- An operating system (OS) is a type of software interface between the user and the device hardware. This software allows users to communicate with the device and perform the desired functions. Operating systems use two components to manage computer programs and applications:
    1. The **kernel** is the core inner component that processes data at the hardware level. It handles input-output management, memory and process management.
    2. The **shell** is the outer layer that manages the interaction between the user and the OS. The shell communicates with the operating system by either taking the input from the user or a shell script. A shell script is a sequence of system commands that are stored in a file.

Batch OS

- The batch operating system does not have a direct link with the computer. A different system divides and allocates similar tasks into batches for easy processing and faster response.
- The batch operating system is appropriate for lengthy and time-consuming tasks. To avoid slowing down a device, each user prepares their tasks offline and submits them to an operator.

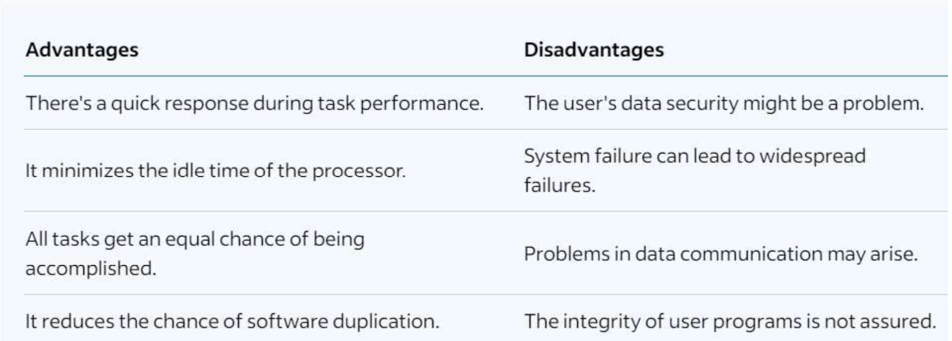
Advantages and Disadvantages of Batch Operating Systems



Advantages	Disadvantages
Many users can share batch systems. There is little idle time for batch operating systems.	Some notable disadvantages are: Batch operating systems are challenging to debug.
It becomes possible to manage large workloads.	Any failure of the system creates a backlog.
It's easy to estimate how long a task will take to be completed.	It may be costly to install and maintain good batch operating systems.

Batch operating systems are used for tasks such as managing payroll systems, data entry and bank statements.

Advantages and disadvantages of time-sharing operating systems:



Advantages	Disadvantages
There's a quick response during task performance.	The user's data security might be a problem.
It minimizes the idle time of the processor.	System failure can lead to widespread failures.
All tasks get an equal chance of being accomplished.	Problems in data communication may arise.
It reduces the chance of software duplication.	The integrity of user programs is not assured.

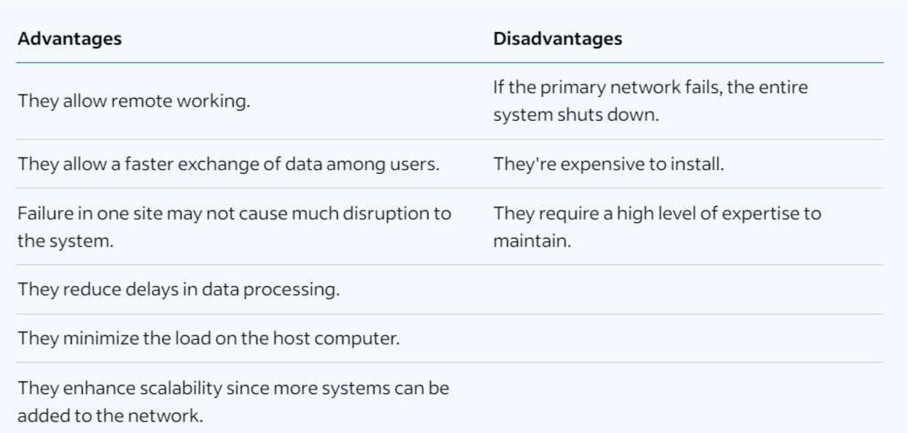
Time-sharing or multitasking OS

- The time-sharing operating system, also known as a multitasking OS, works by allocating time to a particular task and switching between tasks frequently. Unlike the batch system, the time-sharing system allows users to complete their work in the system simultaneously.
- It allows many users to be distributed across various terminals to minimize response time.

Distributed OS

- This system is based on autonomous but interconnected computers communicating with each other via communication lines or a shared network. Each autonomous system has its own processor that may differ in size and function.
- A distributed operating system serves multiple applications and multiple users in real time. The data processing function is then distributed across the processors.

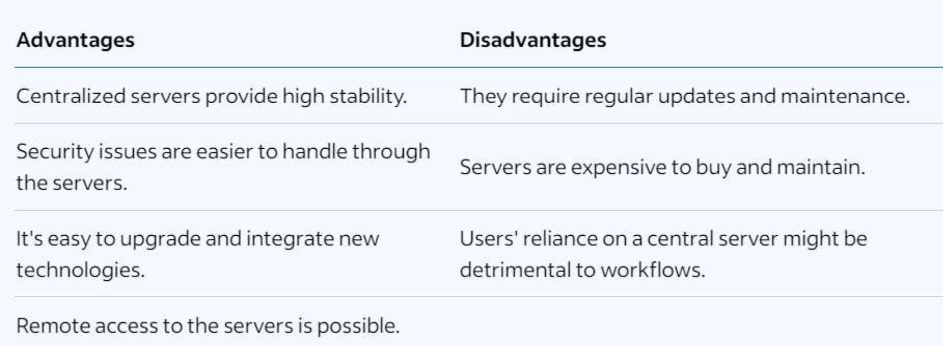
Advantages and disadvantages of distributed operating systems:



Advantages	Disadvantages
They allow remote working.	If the primary network fails, the entire system shuts down.
They allow a faster exchange of data among users.	They're expensive to install.
Failure in one site may not cause much disruption to the system.	They require a high level of expertise to maintain.
They reduce delays in data processing.	
They minimize the load on the host computer.	
They enhance scalability since more systems can be added to the network.	

Distributed operating systems are used for tasks such as telecommunication networks, airline reservation controls and peer-to-peer networks.

Advantages and disadvantages of network operating systems:



Advantages	Disadvantages
Centralized servers provide high stability.	They require regular updates and maintenance.
Security issues are easier to handle through the servers.	Servers are expensive to buy and maintain.
It's easy to upgrade and integrate new technologies.	Users' reliance on a central server might be detrimental to workflows.
Remote access to the servers is possible.	

Examples of network operating systems include Microsoft Windows, Linux and macOS X.

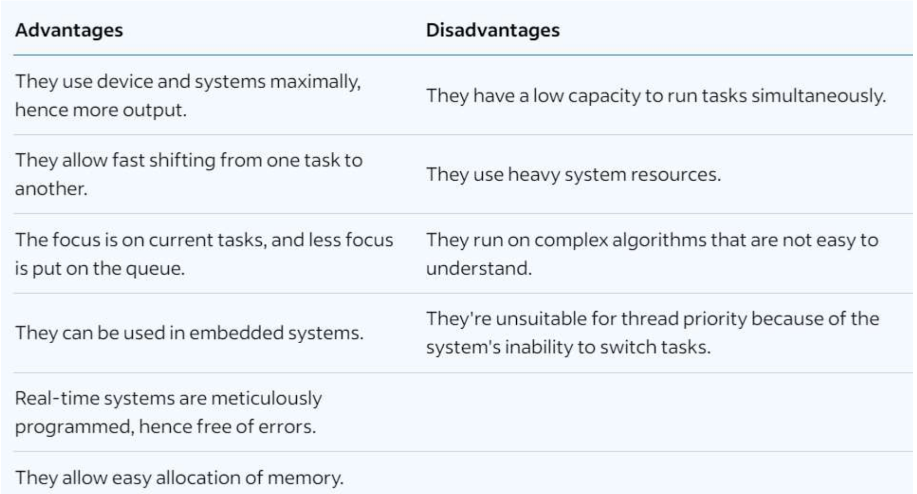
Network OS

- 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.

Real-time OS

- Real-time operating systems provide support to real-time systems that require observance of strict time requirements. The response time between input, processing and response is tiny, which is beneficial for processes that are highly sensitive and need high precision.
- These processes include operating missile systems, medical systems or air traffic control systems, where delays may lead to loss of life and property. Real-time operating systems may either be hard real-time systems or soft real-time systems. Hard real-time systems are installed in applications with strict time constraints.
- The system guarantees the completion of sensitive tasks on time. Hard real-time does not have virtual memory. Soft real-time systems do not have equally rigid time requirements. A critical task gets priority over other tasks.

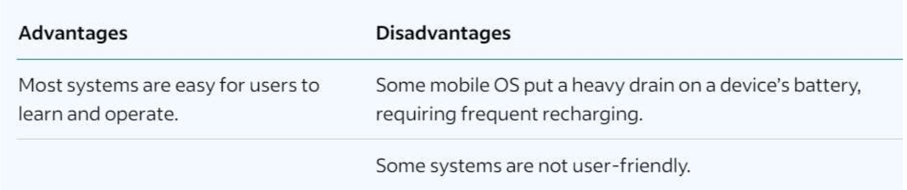
Advantages and disadvantages of real-time operating systems:



Advantages	Disadvantages
They use device and systems maximally, hence more output.	They have a low capacity to run tasks simultaneously.
They allow fast shifting from one task to another.	They use heavy system resources.
The focus is on current tasks, and less focus is put on the queue.	They run on complex algorithms that are not easy to understand.
They can be used in embedded systems.	They're unsuitable for thread priority because of the system's inability to switch tasks.
Real-time systems are meticulously programmed, hence free of errors.	
They allow easy allocation of memory.	

Real-time operating systems are used for tasks such as scientific experiments, medical imaging, robotics and air traffic control operations.

Advantages and disadvantages of mobile operating systems are:



Advantages	Disadvantages
Most systems are easy for users to learn and operate.	Some mobile OS put a heavy drain on a device's battery, requiring frequent recharging.
	Some systems are not user-friendly.

Examples of mobile operating systems include Android OS, Apple and Windows mobile OS.

Mobile OS:

- 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.

Common Operating Systems

- **Microsoft Windows**
- **Apple iOS**
- **Google Android**
- **Apple macOS**
- **Linux**

- **Microsoft Windows:** Created by Microsoft, Microsoft Windows is one of the most popular proprietary operating systems for computers in the world. Most personal computers come preloaded with a version of Microsoft Windows. One downside of Windows is that compatibility with mobile phones has been problematic.
- **Apple iOS:** Apple iOS from Apple is used on smartphones and tablets manufactured by the same company. Users of this system have access to hundreds of applications. The operating system offers strong encryption capabilities to control unauthorized access to users' private data.

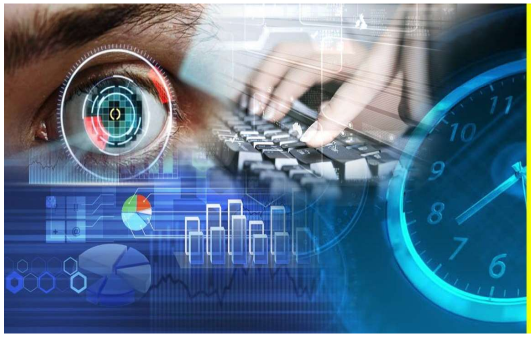
- **Google Android:** Android from Google is the most popular operating system in the world. It's mainly used on tablets and smartphones. It also runs on devices made by other manufacturers. Users have access to numerous mobile applications available on the Google Play Store.
- **Apple macOS:** Developed by Apple, this proprietary operating system runs on the manufacturer's personal computers and desktops. All Apple and Macintosh computers come equipped with the latest version of macOS, previously known as OS X systems. The ability to prevent bugs and fend off hackers make Apple operating systems popular with their users.
- **Linux:** Created by the Finnish programmer Linus Torvalds, Linux is today developed by programmer collaborators across the world who submit tweaks to the central kernel software. Linux is popular with programmers and corporate servers. It is available for free online.

Exercise:

- Give examples for the each type of “Operating Systems” mentioned in the slide number 07.

03. Operating System Concepts.

(Part -3)



HNDIT 3052 Operating Systems

PROCESSES

PROCESSES AND THREADS

- This is actually a detailed study of how operating systems are designed and constructed.
- This is actually going to make a detailed study of how operating systems are **designed and constructed.**
- The most central concept in any operating system is the ***process*:** an abstraction of a running program
- All the other things depend on this concept, and the operating system designer (and student) should have a thorough understanding of what a process is as early as possible.

Topic: PROCESSES AND THREADS

Subtopics:

- The Process Model
- Implementation of Process

Introduction:

- Processes are one of the oldest and most important abstractions that operating systems provide. They support the ability to have (pseudo) concurrent operation even when there is only one CPU available.
- They turn a single CPU into multiple virtual CPUs. Without the process abstraction, modern computing could not exist.
- In this section it is discussed detail about processes and their first relations, threads in the next slide.

Introduction

- Processes are one of the oldest and most important abstractions that operating systems provide. They support the ability to have (pseudo) simultaneous operation even when there is only one CPU available. They turn a single CPU into multiple virtual CPUs. Without the process abstraction, modern computing could not exist.

PROCESSES:

- Process Concept
- Process Scheduling
- Operations on Processes
- Inter-process Communication
- Examples of IPC Systems
- Communication in Client-Server Systems

Process Concept

- An operating system executes a variety of programs:
  - Batch system – **jobs**
  - Time-shared systems – **user programs** or **tasks**
- Textbook uses the terms ***job*** and ***process*** almost interchangeably
- **Process** – a program in execution; process execution must progress in sequential fashion
- Multiple parts
  - The program code, also called **text section**
  - Current activity including **program counter**, processor registers
  - **Stack** containing temporary data
    - Function parameters, return addresses, local variables
  - **Data section** containing global variables
  - **Heap** containing memory dynamically allocated during run time

Process Concept (...)

- Program is ***passive*** entity stored on disk (**executable file**), process is ***active***
  - Program becomes process when executable file loaded into memory
- Execution of program started via GUI mouse clicks, command line entry of its name, etc
- One program can be several processes
  - Consider multiple users executing the same program

Process in Memory

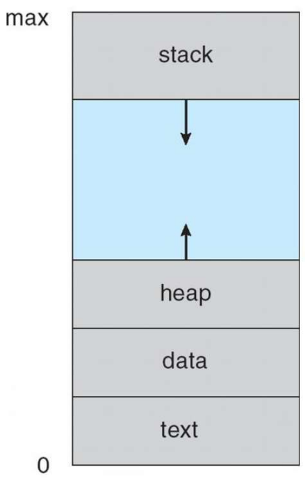
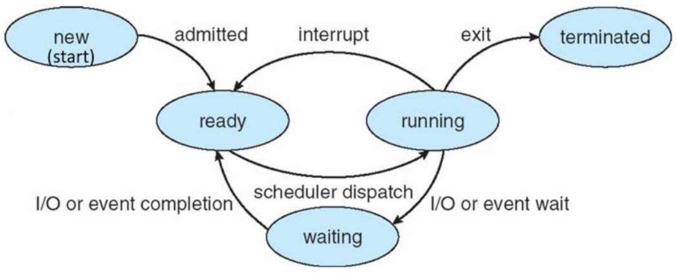


Diagram of Process State (Process life cycle)

- When a process executes, it passes through different states. These stages may differ in different operating systems, and the names of these states are also not standardized.
- In general, a process can have one of the following five states at it's lice cycle time.



Process State

- As a process executes, it changes **state**
  - **new**: The process is being created
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **ready**: The process is waiting to be assigned to a processor
  - **terminated**: The process has finished execution

Process state /Life Cycle

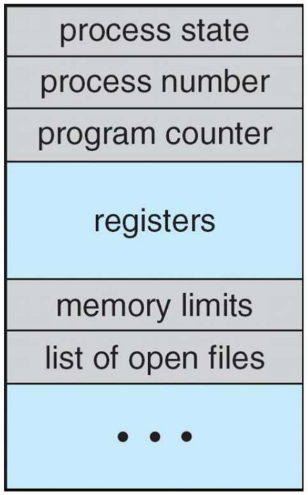
S/N	State & Description
1.	New/ Start This is the initial state when a process is first started/created.
2.	Ready The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after Start state or while running it by but interrupted by the scheduler to assign CPU to some other process.
3.	Running Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.
4.	Waiting Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.
5.	Terminated or Exit Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.

Process Control Block (PCB)

Information associated with each process

(also called **task control block**)

- Process state – running, waiting, etc
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files



process state

process number

program counter

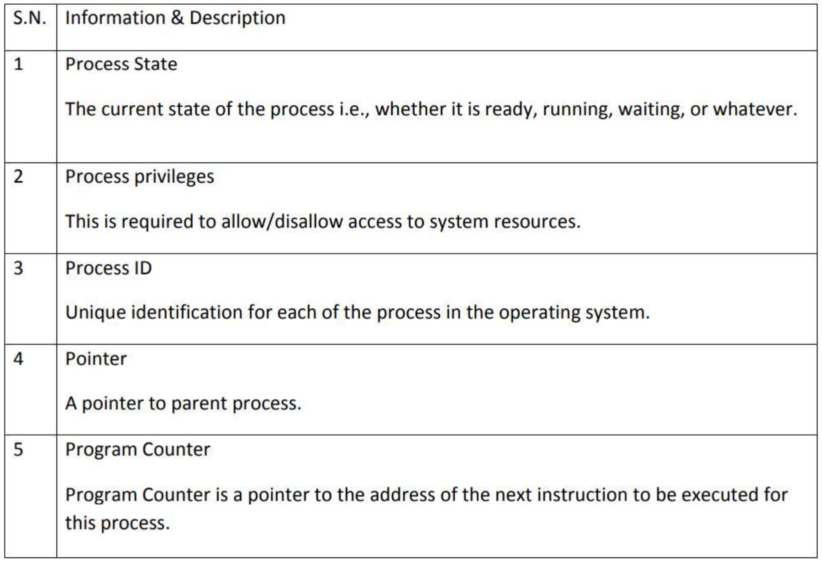
registers

memory limits

list of open files

• • •

PCB...

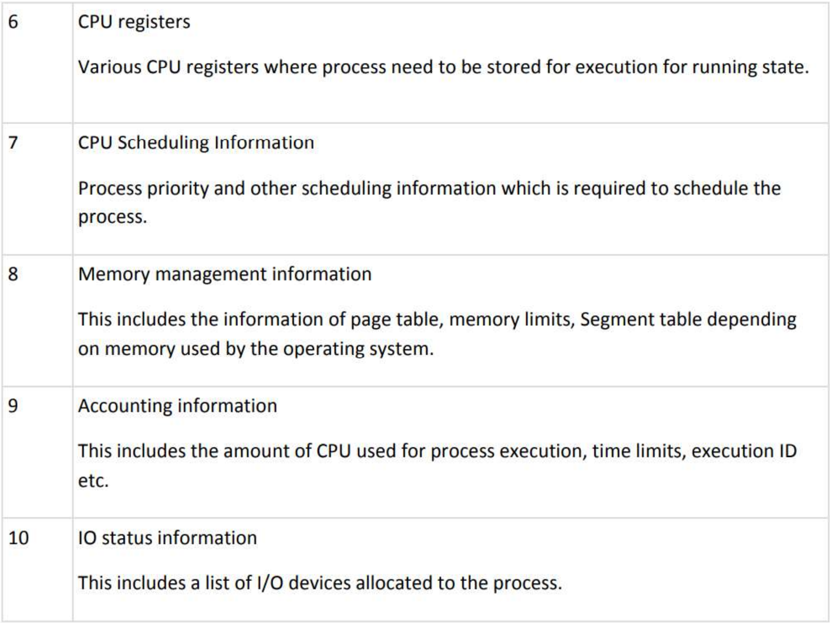


S.N.	Information & Description
1	Process State The current state of the process i.e., whether it is ready, running, waiting, or whatever.
2	Process privileges This is required to allow/disallow access to system resources.
3	Process ID Unique identification for each of the process in the operating system.
4	Pointer A pointer to parent process.
5	Program Counter Program Counter is a pointer to the address of the next instruction to be executed for this process.

PCB..

- A Process Control Block is a data structure maintained by the Operating System for every process. The PCB is identified by an integer process ID (PID). A PCB keeps all the information needed to keep track of a process as listed below in the table in next slide–

PCB..



6	CPU registers Various CPU registers where process need to be stored for execution for running state.
7	CPU Scheduling Information Process priority and other scheduling information which is required to schedule the process.
8	Memory management information This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
9	Accounting information This includes the amount of CPU used for process execution, time limits, execution ID etc.
10	IO status information This includes a list of I/O devices allocated to the process.

- When an operating system is booted, normally various processes are created. Some of these are foreground processes, that is, processes that interact with (human) users and perform work for them. Others are background processes, which are not associated with specific users, but instead have some particular function. For instance, one background process may be designed to accept incoming e-mail, sleeping most of the day but suddenly springing to life when incoming e-mail arrives. Another background process may be designed to accept incoming requests for Web pages hosted on that machine, waking up when a request arrives to service the request. Processes that stay in the background to handle some activity such as e-mail, Web pages, news, printing, and so on are called daemons. Large systems generally have dozens of them. In UNIX, the ps program can be used to list the running processes. In Windows, the task manager can be used.

Process Creation:

- Operating systems require some way to make processes. In very simple systems, or in systems designed for running only a single application (e.g., the controller in a microwave oven), it may be possible to have all the processes that will ever be needed be present when the system comes up. In general-purpose systems, however, some way is required to create and finish processes as required during operation. We will now look at some of the issues.
- There are four principal events that cause processes to be created:

  1 . System initialization.

  2 . Execution of a process creation system call by a running process.

  3 . A user request to create a new process.

  4 . Initiation of a batch job.

Process Termination:

- After a process has been created, it starts running and performs whatever its job is. However, nothing lasts forever, not even processes. Sooner or later the new process will end, generally due to one of the following conditions:

  1. Normal exit (voluntary).
  2. Error exit (voluntary).
  3. Fatal error (involuntary).
  4. Killed by another process (involuntary).

Process Hierarchies:

- In some systems, when a process creates another process, the parent process and child process continue to be connected in certain ways. The child process can itself create more processes, forming a process hierarchy. Note that unlike plants and animals that use sexual reproduction, a process has only one parent (but zero, one, two, or more children).
- In UNIX, a process and all of its children and further descendants together form a process group. When a user sends a signal from the keyboard, the signal is delivered to all members of the process group currently connected with the keyboard (generally all active processes that were created in the current window). Individually, each process can catch the signal, ignore the signal, or take the default action, which is to be killed by the signal.

Implementation of Processes:

- To implement the process model, the operating system maintains a table (an array of structures), called the process table, with one entry per process. (Some authors call these entries process control blocks.)
- This entry includes important information about the process state, containing its program counter, stack pointer, memory allocation, the status of its open files, its accounting and scheduling information, and everything else about the process that must be saved when the process is switched from running to ready or blocked state so that it can be restarted later as if it had never been stopped.

Implementation of Processes…

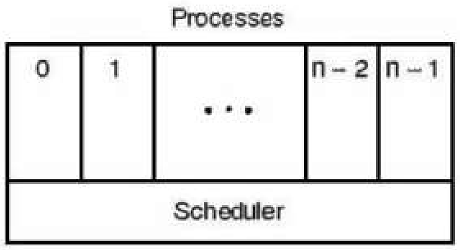


Figure (a). The lowest layer of a process-structured operating system handles interrupts and scheduling. Above that layer are sequential processes.

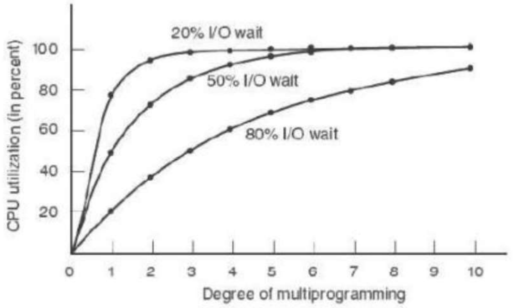
Modeling Multiprogramming:

- When multiprogramming is used, the CPU utilization can be improved. Crudely put, if the average process computes only 20% of the time it is sitting in memory, with five processes in memory at once, the CPU should be busy all the time. This model is unrealistically hopeful, however, since it tacitly assumes that all five processes will never be waiting for I/O at the same time.
- A better model is to look at CPU usage from a probabilistic viewpoint. Assume that a process spends a fraction p of its time waiting for I/O to complete. With n processes in memory at once, the probability that all n processes are waiting for I/O (in which case the CPU will be idle) is p^n .

- The CPU utilization is then given by the formula:

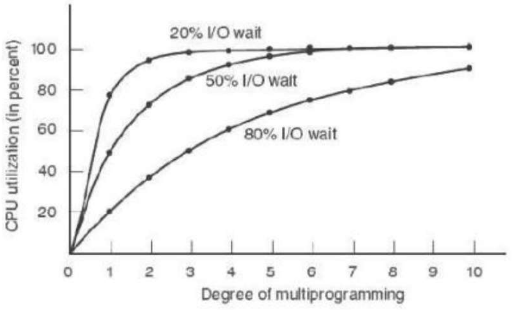
$$\text{CPU utilization} = 1 – p^n$$

- The following figure shows the CPU utilization as a function of n, which is called the degree of multiprogramming.



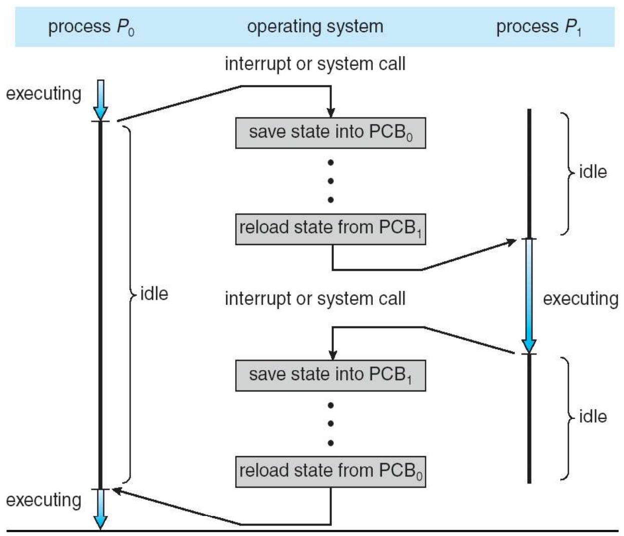
CPU utilization as a function of the number of processes in memory

- From the figure it is clear that if processes spend 80% of their time waiting for I/O, at least 10 processes must be in memory at once to get the CPU waste below 10%. When you understand that an interactive process waiting for a user to type something at a terminal is in I/O wait state, it should be clear that I/O wait times of 80% and more are not abnormal. But even on servers, processes doing a lot of disk I/O will sometimes have this percentage or more.

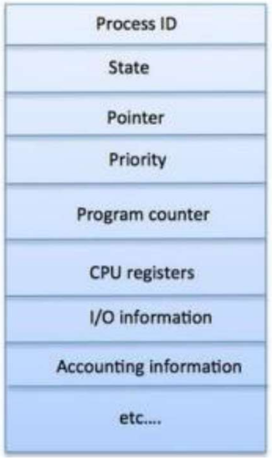


CPU utilization as a function of the number of processes in memory

CPU Switch From Process to Process



The architecture of a PCB is completely dependent on Operating System and may contain different information in different operating systems. Here is a simplified



Process ID

State

Pointer

Priority

Program counter

CPU registers

I/O information

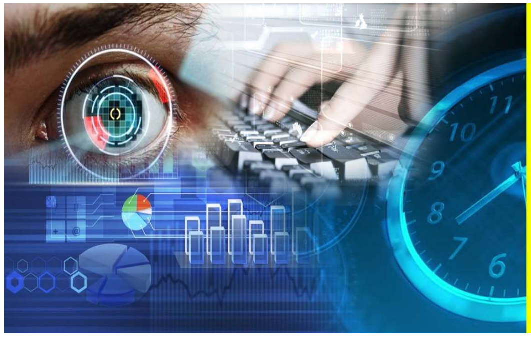
Accounting information

etc....

The diagram of a PCB

NEXT Lesson:

- THREADS



HNDIT 3052 Operating Systems

Threads

Threads

- **What is a Thread?**
- Let us take an example of a human body. A human body has different parts having different functionalities which are working parallelly (Eg: *Eyes, ears, hands*, etc). Similarly in computers, a single process might have multiple functionalities running parallelly where each functionality can be considered as a thread.

Threads

Topic: Threads

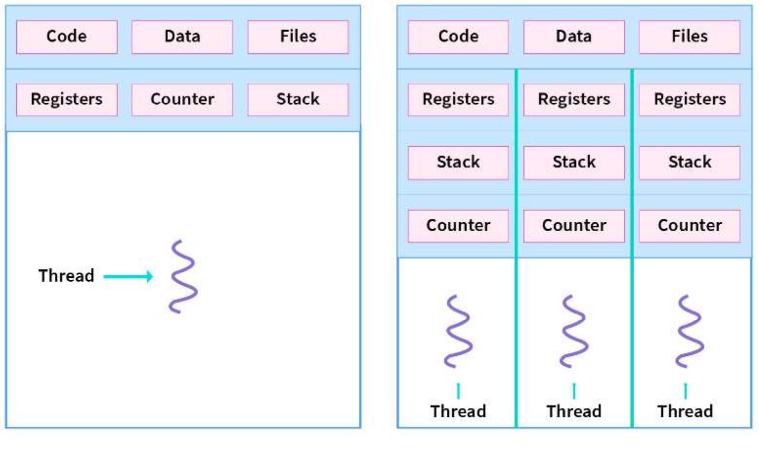
Subtopics:

- The Thread Model
- POSIX Threads
- Implementation of Thread

What is Thread in OS?

- **Thread is a sequential flow of tasks within a process.** Threads in OS can be of the same or different types. Threads are used to increase the performance of the applications.
- Each thread has its own program counter, stack, and set of registers. But the threads of a single process might share the same code and data/file. **Threads are also termed as lightweight processes as they share common resources**.
- **Eg:** While playing a movie on a device the audio and video are controlled by different threads in the background.

Eg: While playing a movie on a device the audio and video are controlled by different threads in the background.



Single-threaded process

Multithreaded process

The above diagram shows the **difference between a single-threaded process and a multithreaded process** and the resources that are shared **among threads** in a multithreaded process.

Why do we need Threads?

Threads in the operating system provide multiple benefits and improve the overall performance of the system. Some of the reasons threads are needed in the operating system are:

- Since threads use the same data and code, the operational cost between threads is low.
- Creating and terminating a thread is faster compared to creating or terminating a process.
- **Context switching** is faster in threads compared to processes.

Components of Thread

A thread has the following three components:

1. Program Counter
2. Register Set
3. Stack space

Context Switching:

- A context switch is a procedure that a computer’s CPU (central processing unit) follows to change from one task (or process) to another while ensuring that the tasks do not conflict.
- Effective context switching is critical if a computer is to provide user-friendly multitasking.

- In a CPU, the term "context" refers to the data in the **registers** and program counter at a specific moment in time. A register holds the current CPU **instruction**. A program counter, also known as an instruction address register, is a small amount of fast **memory** that holds the address of the instruction to be executed immediately after the current one.

Context Switching: …

- A context switch can be performed entirely in **hardware** (physical media). Older CPUs, such as those in the **x86** series, do it that way.
- However, most modern CPUs perform context switches by means of **software** (programming). A modern CPU can perform hundreds of context switches per second.
- Therefore, the user gets the impression that the computer is performing multiple tasks in a **parallel** fashion, when the CPU actually alternates or rotates between or among the tasks at a high rate of speed.

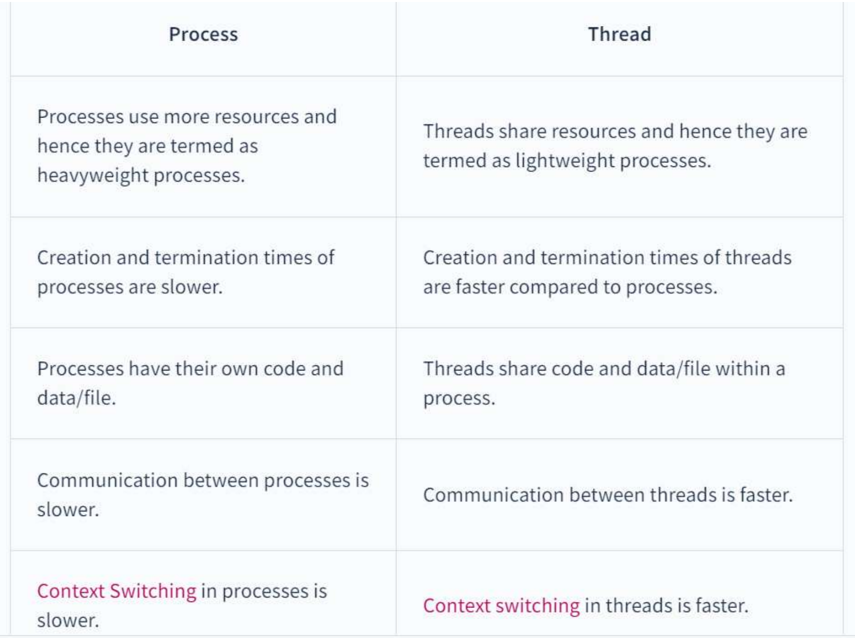
Process vs Thread

- **Process simply means any program in execution while the thread is a segment of a process**. The main differences between process and thread are mentioned below:

Why Multithreading?

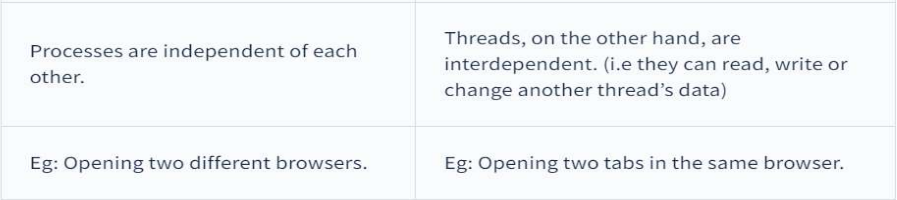
In Multithreading, the idea is to divide a single process into multiple threads instead of creating a whole new process. Multithreading is done to achieve parallelism and to improve the performance of the applications as it is faster in many ways which were discussed above. The other advantages of multithreading are mentioned below.

- **Resource Sharing:** Threads of a single process share the same resources such as code, data/file.
- **Responsiveness:** Program responsiveness enables a program to run even if part of the program is blocked or executing a lengthy operation. Thus, increasing the responsiveness to the user.
- **Economy:** It is more economical to use threads as they share the resources of a single process. On the other hand, creating processes is expensive.

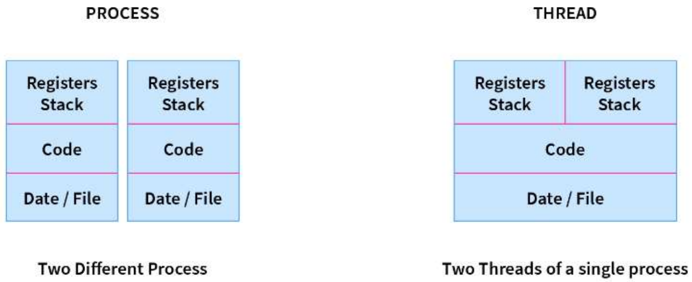


Process	Thread
Processes use more resources and hence they are termed as heavyweight processes.	Threads share resources and hence they are termed as lightweight processes.
Creation and termination times of processes are slower.	Creation and termination times of threads are faster compared to processes.
Processes have their own code and data/file.	Threads share code and data/file within a process.
Communication between processes is slower.	Communication between threads is faster.
Context Switching in processes is slower.	Context switching in threads is faster.

Process vs. Threads...

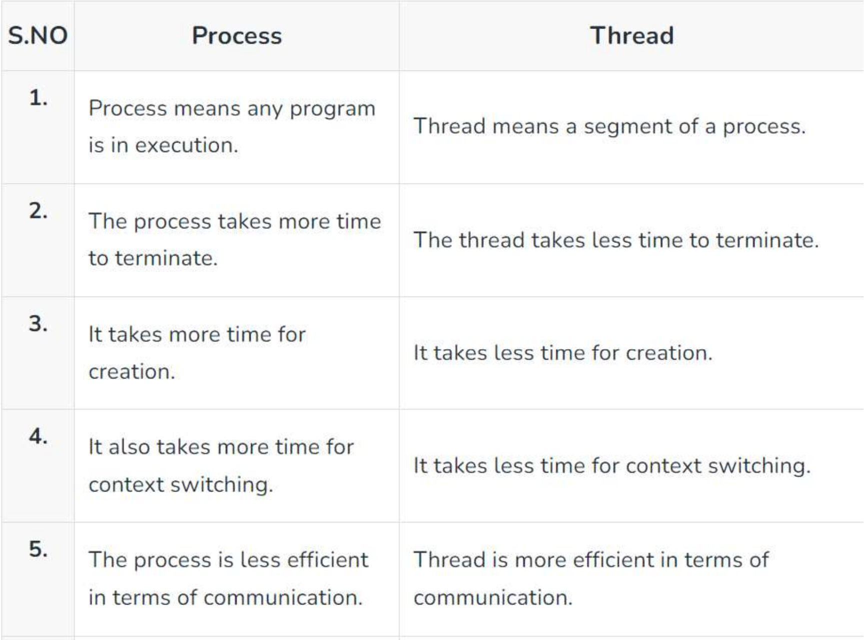


Processes are independent of each other.	Threads, on the other hand, are interdependent. (i.e they can read, write or change another thread's data)
Eg: Opening two different browsers.	Eg: Opening two tabs in the same browser.

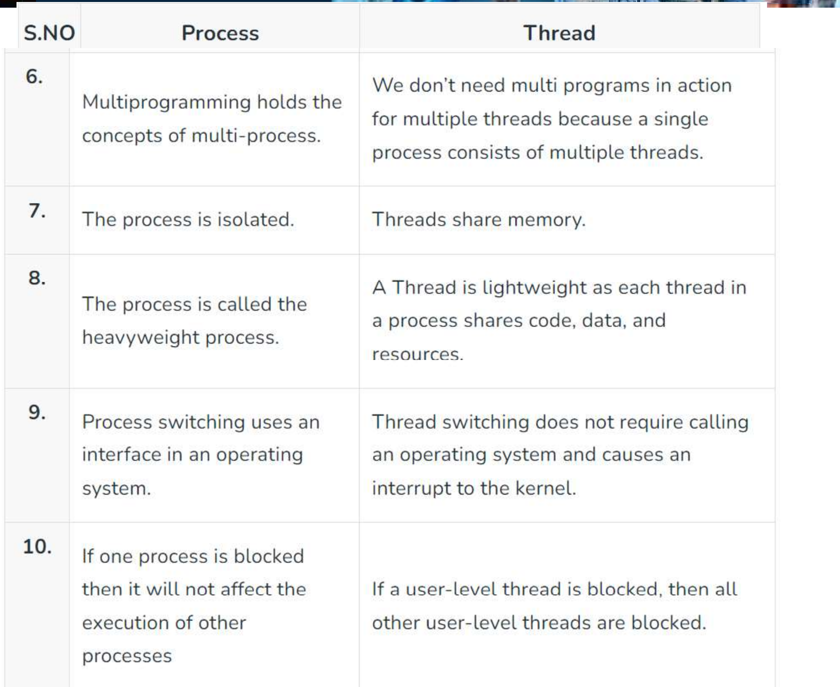


The above diagram shows how the resources are shared in two different processes vs two threads in a single process.

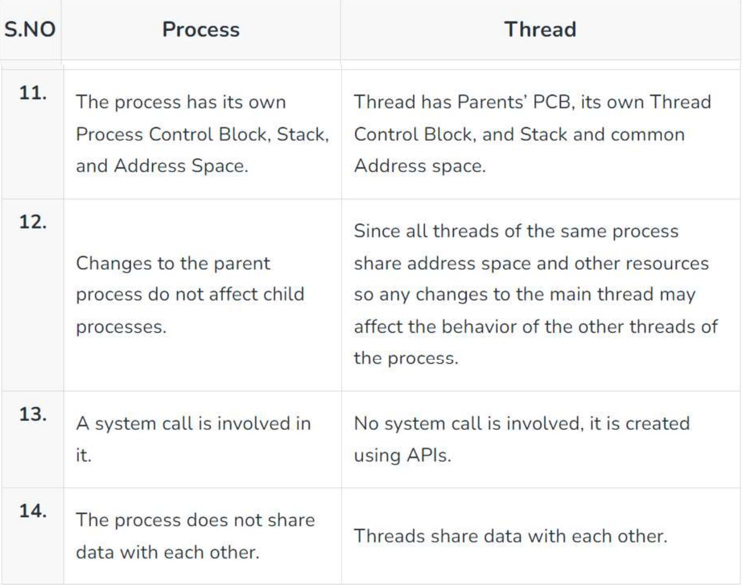
Another Different comparison of Thread & Process.



S.NO	Process	Thread
1.	Process means any program is in execution.	Thread means a segment of a process.
2.	The process takes more time to terminate.	The thread takes less time to terminate.
3.	It takes more time for creation.	It takes less time for creation.
4.	It also takes more time for context switching.	It takes less time for context switching.
5.	The process is less efficient in terms of communication.	Thread is more efficient in terms of communication.



S.NO	Process	Thread
6.	Multiprogramming holds the concepts of multi-process.	We don't need multi programs in action for multiple threads because a single process consists of multiple threads.
7.	The process is isolated.	Threads share memory.
8.	The process is called the heavyweight process.	A Thread is lightweight as each thread in a process shares code, data, and resources.
9.	Process switching uses an interface in an operating system.	Thread switching does not require calling an operating system and causes an interrupt to the kernel.
10.	If one process is blocked then it will not affect the execution of other processes	If a user-level thread is blocked, then all other user-level threads are blocked.



S.NO	Process	Thread
11.	The process has its own Process Control Block, Stack, and Address Space.	Thread has Parents' PCB, its own Thread Control Block, and Stack and common Address space.
12.	Changes to the parent process do not affect child processes.	Since all threads of the same process share address space and other resources so any changes to the main thread may affect the behavior of the other threads of the process.
13.	A system call is involved in it.	No system call is involved, it is created using APIs.
14.	The process does not share data with each other.	Threads share data with each other.

Note: *In some cases where the thread is processing a bigger workload compared to a process's workload then the thread may take more time to terminate. But this is an extremely rare situation and has fewer chances to occur.*

Types of Thread

- **1. User Level Thread:**
- **2. Kernel level Thread:**

1. User Level Thread:

User-level threads are implemented and managed by the user and the kernel is not aware of it.

- User-level threads are **implemented using user-level libraries and the OS does not recognize these threads**.
- User-level thread is **faster to create and manage compared to kernel-level thread**.
- **Context switching in user-level threads is faster**.
- If one user-level thread performs a blocking operation then the entire process gets blocked.
  - Eg: POSIX threads, Java threads, etc.

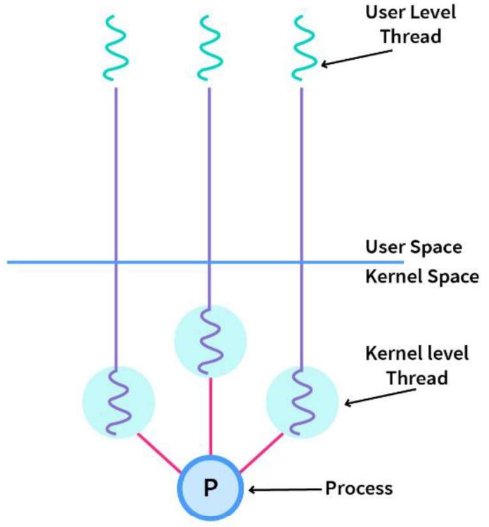
User Threads and Kernel Threads

1. **User threads** - management done by user-level threads library.
   - Three primary thread libraries:
     - POSIX **Pthreads**
     - Windows threads
     - Java threads
2. **Kernel threads** - Supported by the Kernel
   - Examples – virtually all general purpose operating systems, including:
     - Windows
     - Solaris
     - Linux
     - Tru64 UNIX
     - Mac OS X

2. Kernel level Thread:

Kernel level threads are implemented and managed by the OS.

- Kernel level threads are **implemented using system calls and Kernel level threads are recognized by the OS**.
- Kernel-level threads are **slower to create and manage compared to user-level threads**.
- **Context switching in a kernel-level thread is slower**.
- Even if one kernel-level thread performs a blocking operation, it does not affect other threads. Eg: **Window Solaris**.



The above diagram shows the functioning of user-level threads in **user space** and kernel-level threads in **kernel space**.

Issues with Threading

There are a number of issues that arise with threading. Some of them are mentioned below:

- **The semantics of fork() and exec() system calls: The fork() call is used to create a duplicate child process**.
- During a fork() call the issue that arises is whether the whole process should be duplicated or just the thread which made the fork() call should be duplicated.
- **The exec() call replaces the whole process that called it including all the threads in the process with a new program**.

Advantages of Threading:

1. Threads improve the overall performance of a program.
2. Threads increases the responsiveness of the program
3. Context Switching time in threads is faster.
4. Threads share the same memory and resources within a process.
5. Communication is faster in threads.
6. Threads provide concurrency within a process.
7. Enhanced throughput of the system.
8. Since different threads can run parallelly, threading enables the utilization of the multiprocessor architecture to a greater extent and increases efficiency.

Issues with Threading…

- **Thread cancellation:** The termination of a thread before its completion is called thread cancellation and the terminated thread is termed as target thread. Thread cancellation is of two types:
  - **Asynchronous Cancellation:** In asynchronous cancellation, one thread immediately terminates the target thread.
  - **Deferred Cancellation:** In deferred cancellation, the target thread periodically checks if it should be terminated.

Issues with Threading...

- **Signal handling:** In UNIX systems, a signal is used to notify a process that a particular event has happened. Based on the source of the signal, signal handling can be categorized as:
  - **Asynchronous Signal:** The signal which is generated outside the process which receives it.
  - **Synchronous Signal:** The signal which is generated and delivered in the same process.

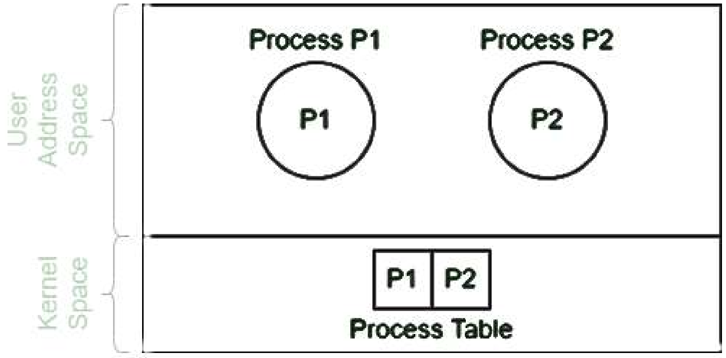
The Thread Model

Thread Models in Operating System:

- A thread is a light weight process which is similar to a process where every process can have one or more threads. Each thread contains a Stack and a Thread Control Block. There are four basic thread models :
  1. User Level Single Thread Model :
  2. User Level Multi Thread Model :
  3. Kernel Level Single Thread Model :
  4. Kernel Level Multi Thread Model **:**

1. User Level Single Thread Model :

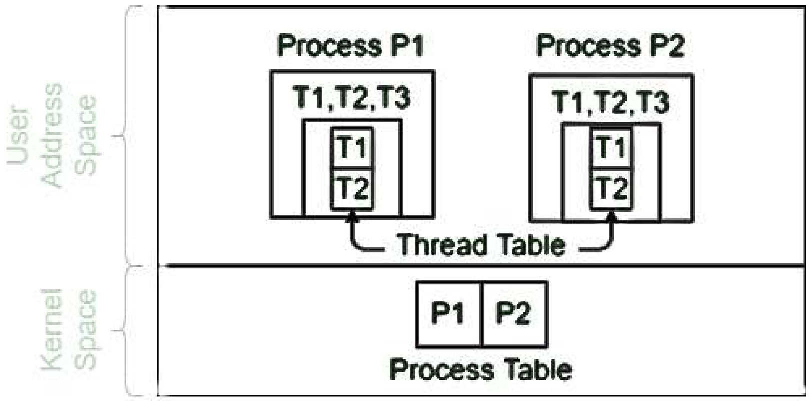
- Each process contains a single thread.
- Single process is itself a single thread.
- process table contains an entry for every process by maintaining its PCB.



2. User Level Multi Thread Model :

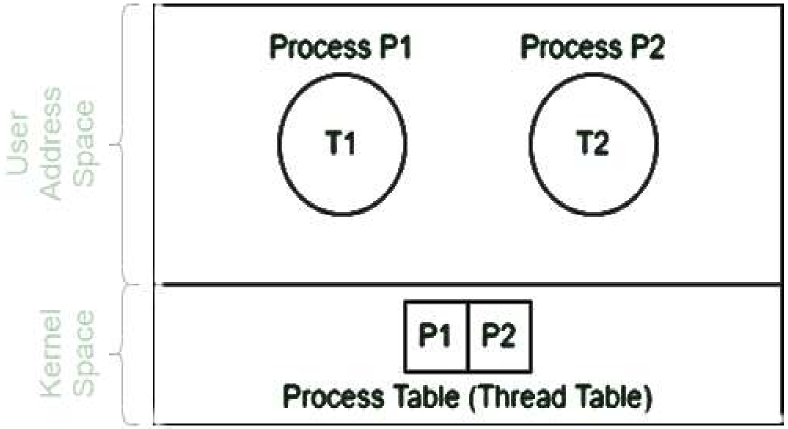
- Each process contains multiple threads.
- All threads of the process are scheduled by a thread library at user level.
- Thread switching can be done faster than process switching.
- Thread switching is independent of operating system which can be done within a process.
- Blocking one thread makes blocking of entire process.
- Thread table maintains Thread Control Block of each thread of a process.
- Thread scheduling happens within a process and not known to Kernel.

2. User Level Multi Thread Model ...



3. Kernel Level Single Thread Model :

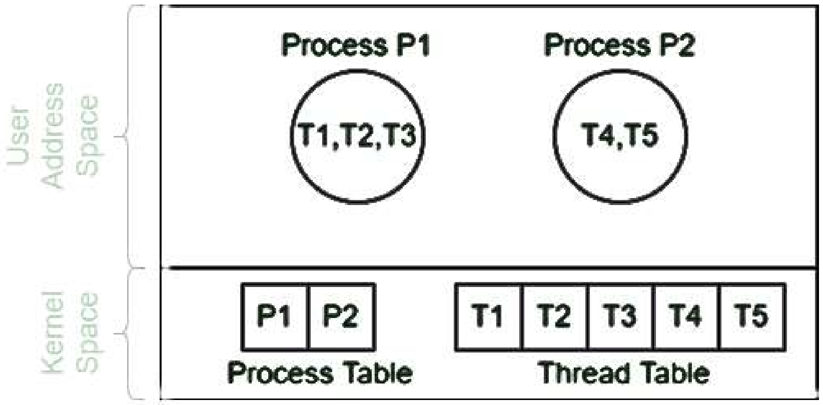
- Each process contains a single thread.
- Thread used here is kernel level thread.
- Process table works as thread table.



4. Kernel Level Multi Thread Model :

- Thread scheduling is done at kernel level.
- Fine grain scheduling is done on a thread basis.
- If a thread blocks, another thread can be scheduled without blocking the whole process.
- Thread scheduling at Kernel process is slower compared to user level thread scheduling.
- Thread switching involves switch.

4. Kernel Level Multi Thread Model ...

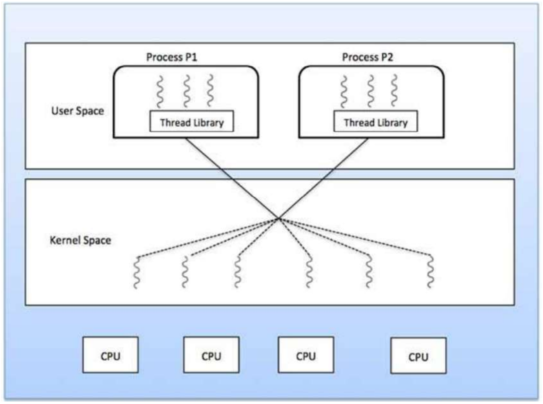


Multithreading Models

- Some operating system provide a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process. Multithreading models are three types.
  1. Many to many relationship.
  2. Many to one relationship.
  3. One to one relationship.

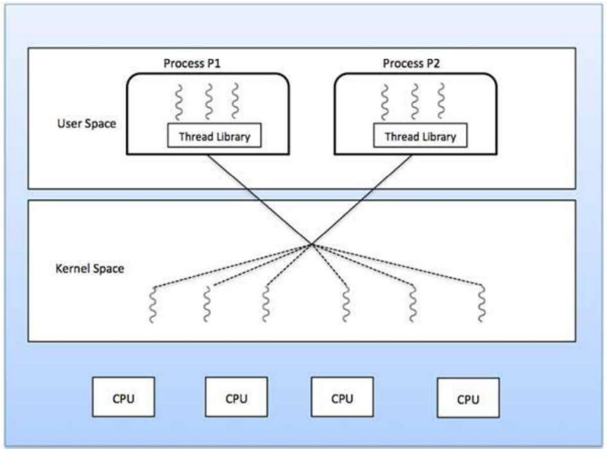
i. Many to Many Multithreading Model...

- The following diagram shows the many-to-many threading model where 6 user level threads are multiplexing with 6 kernel level threads. In this model, developers can create as many user threads as necessary and the corresponding Kernel threads **can run in parallel on a multiprocessor machine.** This model **provides the best accuracy on concurrency** and when a **thread** performs a blocking system call, the **kernel** can schedule another thread for execution.



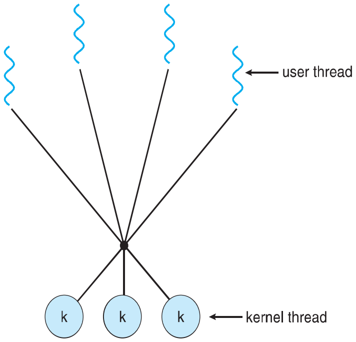
i. Many to Many Multithreading Model:

- The many-to-many model **multiplexes** any number of user threads onto an equal or smaller number of kernel threads.



i. Many to Many Multithreading Model...

- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- Solaris prior to version 9
- Windows with the *ThreadFiber* package.
- Fiber:
  - A *fiber* is a unit of execution that must be manually scheduled by the application.
  - Fibers run in the context of the threads that schedule them.
  - Each thread can schedule multiple fibers. In general, fibers do not provide advantages over a well-designed multithreaded application.
  - However, using fibers can make it easier to port applications that were designed to schedule their own threads.

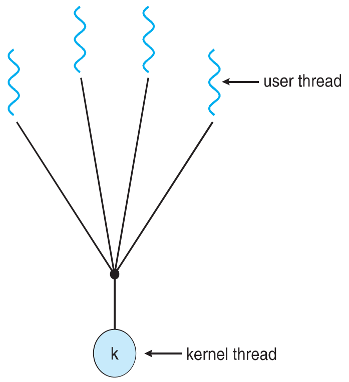


ii. Many to One Multithreading Model

- Many-to-one model maps many user level threads to one Kernel-level thread. Thread management is done in user space by the thread library. When thread makes a blocking system call, the entire process will be blocked. Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.

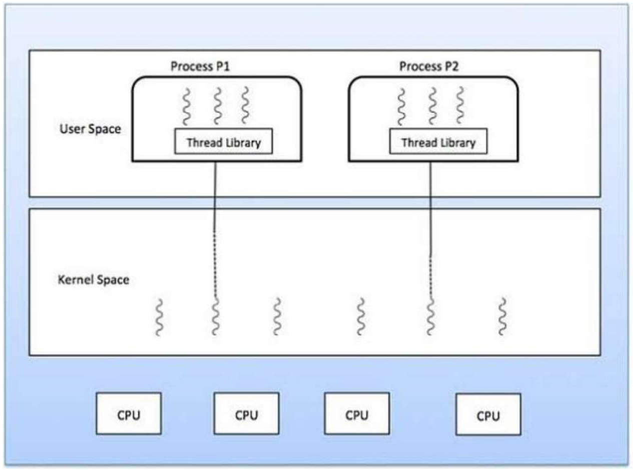
ii. Many to One Multithreading Model…

- Many user-level threads mapped to single kernel thread.
- One thread blocking causes all to block.
- Multiple threads may not run in parallel on multicore system because only one may be in kernel at a time.
- Few systems currently use this model.
- Examples:
  - Solaris Green Threads
  - GNU Portable Threads



ii. Many to One Multithreading Model…

- If the user-level thread libraries are implemented in the operating system in such a way that the system does not support them, then the Kernel threads use the many-to-one relationship modes.

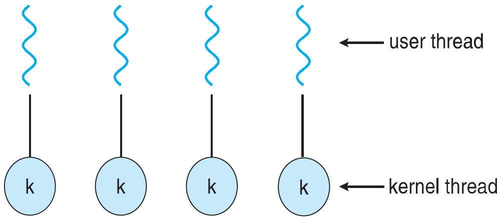


iii. One to One Multithreading Model

- There is one-to-one relationship of user-level thread to the kernel-level thread. This model provides more concurrency than the many-to-one model. It also allows another thread to run when a thread makes a blocking system call. It supports multiple threads to execute in parallel on microprocessors.

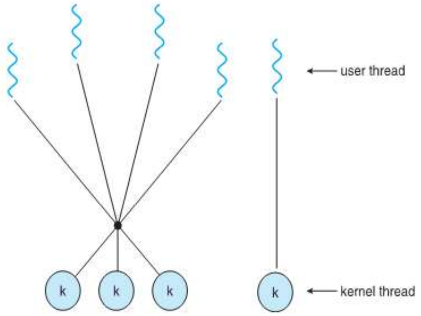
iii. One to One Multithreading Model…

- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Number of threads per process sometimes restricted due to overhead
- Examples
  - Windows
  - Linux
  - Solaris 9 and later



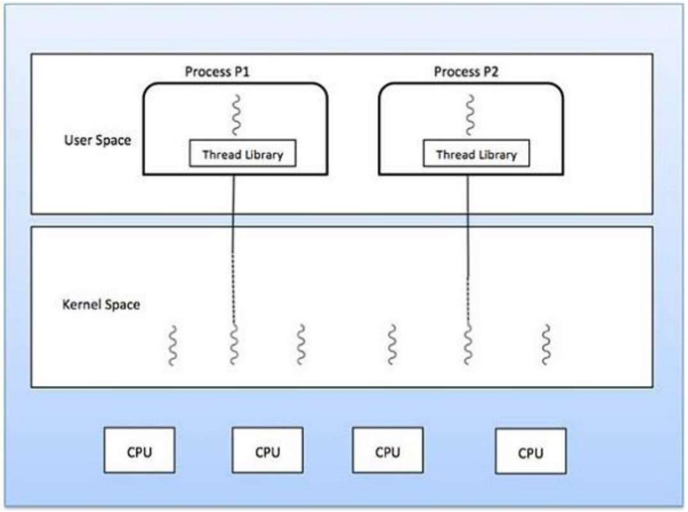
(iv). Two-level Model (similar ti many to many model, but:)

- Similar to M:M, except that **it allows a user thread to be bound to kernel thread**
- Examples
  - IRIX
  - HP-UX
  - Tru64 UNIX
  - Solaris 8 and earlier



iii. One to One Multithreading Model…

- Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2, windows NT and windows 2000 use one to one relationship model.



How multiprogramming of processes works?

- When a multithreaded process is run on a single-CPU system, the threads take turns running.
- In Fig. 2-1 (next slide) , we saw how multiprogramming of processes works.
- By switching back and forth among multiple processes, the system gives the illusion of separate sequential processes running in parallel.
- Multithreading works the same way. The CPU switches rapidly back and forth among the threads, providing the illusion that the threads are running in parallel, albeit on a slower CPU than the real one. With three compute-bound threads in a process, the threads would appear to be running in parallel, each one on a CPU with one-third the speed of the real CPU.

How multiprogramming of processes works?

- Different threads in a process are not as independent as different processes. All threads have exactly the same address space, which means that they also share the same global variables. Since every thread can access every memory address within the process' address space, one thread can read, write, or even wipe out another thread's stack.
- There is no protection between threads because
  - (1) it is impossible, and
  - (2) it should not be necessary.

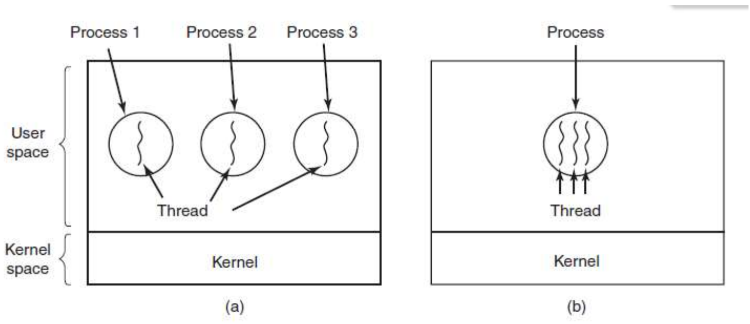


Figure 2-11. (a) Three processes each with one thread. (b) One process with three threads.

Each thread has its own stack. Give the reason...

- **Example:**
  - If procedure *X* calls procedure *Y* and,
  - *Y* calls procedure *Z*, then
  - while *Z* is executing,
  - the frames for *X*, *Y*, and *Z* will all be on the stack.
- **Each thread** will generally call different procedures and thus have a different execution history.
- This is why each thread needs its own stack.

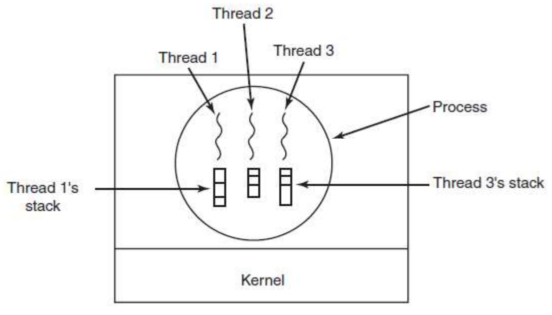


Figure 2-13. Each thread has its own stack.

Each thread has its own stack. Give the reason.

- **Each thread** has its **own stack**, as illustrated in Fig. 2-13 in next slide.
- **Each thread's stack** contains **one frame** for each procedure called but not yet returned from.
- **This frame contains** the **procedure's local variables** and the **return address** to use when the procedure call has finished.
- **Example:**
  - If procedure *X* calls procedure *Y* and,
  - *Y* calls procedure *Z*, then
  - while *Z* is executing,
  - the frames for *X*, *Y*, and *Z* will all be on the stack.
- **Each thread** will generally call different procedures and thus have a different execution history.
- This is why each thread needs its own stack.

Thread Libraries

- **Thread library** provides programmer with API for creating and managing threads
- Two primary ways of implementing
  - Library entirely in user space
  - Kernel-level library supported by the OS

POSIX Threads (Pthreads)

- May be provided either as user-level or kernel-level
- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- ***Specification***, not ***implementation***
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX operating systems (Solaris, Linux, Mac OS X)

#include <pthread.h>
#include <pthread.h>
#include <pthread.h>

POSIX Threads in OS :

- To utilise the PThread interfaces, we must include the header pthread.h at the start of the CPP script.

```
#include <pthread.h>
```

- PThreads is a highly concrete multithreading system that is the UNIX system's default standard.
- **PThreads** is an abbreviation for **POSIX threads**, and POSIX is an abbreviation for **Portable Operating System Interface**, which is a type of interface that the operating system must implement.
- PThreads in POSIX outline the **threading APIs** that the operating system must provide.

POSIX Threads in OS :

- The POSIX thread libraries are a C/C++ thread API based on standards. It enables the creation of a new concurrent process flow. It works well on multi-processor or multi-core systems, where the process flow may be scheduled to execute on another processor, increasing speed through parallel or distributed processing. Because the system does not create a new system, virtual memory space and environment for the process, threads needless overhead than "forking" or creating a new process. While multiprocessor systems are the most effective, benefits can also be obtained on uniprocessor systems that leverage delay in I/O and other system processes that may impede process execution.

Why is Pthreads used?

- The fundamental purpose for adopting **Pthreads** is to improve programme performance.
- When compared to the expense of starting and administering a process, a thread requires far less operating system overhead. Thread management takes fewer system resources than process management.
- A process's threads all share the same address space. Inter-thread communication is more efficient and, in many circumstances, more user-friendly than inter-process communication.
- Threaded applications provide possible performance increases and practical advantages over non-threaded programmes in a variety of ways.
- Multi-threaded programmes will run on a single-processor system but will automatically make use of a multiprocessor machine without the need for recompilation.
- The most significant reason for employing **Pthreads** in a multiprocessor system is to take advantage of possible parallelism.
- In order for a programme to use **Pthreads**, it must be divided into discrete, independent tasks that may run concurrently.

Implementation of Thread

- There are two ways to implement a thread, they're either in user space or in the Kernel.
- The corresponding code and the data structures used are stored in the user space. If an API is invoked, it results in a local system call in user space, rather than a system call.

Implementation of User-level Threads

Implementation at user-level

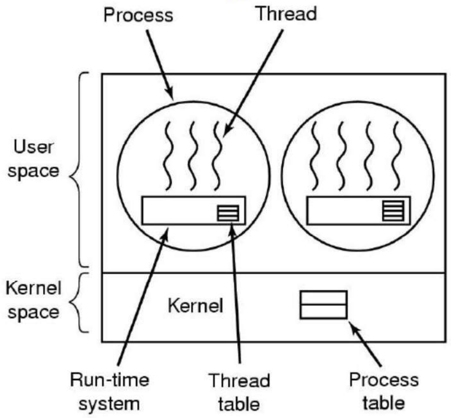
– User-level Thread Control Block (TCB), ready queue, blocked queue, and dispatcher.

– Kernel has no knowledge of the threads (it only sees a single process).

– If a thread blocks waiting for a resource held by another thread, its state is save and the dispatcher switches to another ready thread.

– Thread management (create, exit, yield, wait) are implemented in a runtime support library.

Implementing Threads in User Space



A user-level threads package

Pros - User-level Threads

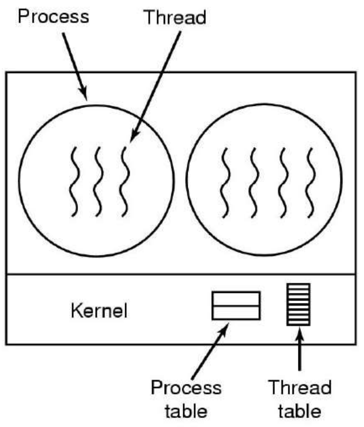
- Pros
  - Thread management and switching at user level is much faster than doing it in kernel level
    - No need to trap into kernel and back to switch
  - Dispatcher algorithm can be tuned to the application
    - E.g. use priorities
  - Can be implemented on any OS (thread or non-thread aware)
  - Can easily support massive numbers of threads on a per-application basis
    - Use normal application virtual memory
    - Kernel memory more constrained. Difficult to efficiently support wildly differing numbers of threads for different applications.

Cons - User-level Threads

User-level Threads

- Cons
  - Threads have to yield() manually (no timer interrupt delivery to user-level)
    - Co-operative multithreading
      - A single poorly design/implemented thread can monopolise the available CPU time
    - There are work-arounds (e.g. a timer signal per second to enable pre-emptive multithreading), they are course grain and a kludge.
  - Does not take advantage of multiple CPUs (in reality, we still have a single threaded process as far as the kernel is concerned)

Implementing Threads in the Kernel



A threads package managed by the kernel

Cons - User-level Threads

- Cons
  - If a thread makes a blocking system call (or takes a page fault), the process (and all the internal threads) blocks
    - Can't overlap I/O with computation
    - Can use wrappers as a work around
      - Example: wrap the `read()` call
      - Use `select()` to test if read system call would block
        - » `select()` then `read()`
        - » Only call `read()` if it won't block
        - » Otherwise schedule another thread
      - Wrapper requires 2 system calls instead of one
        - » Wrappers are needed for environments doing lots of blocking system calls?
    - Can change to kernel to support non-blocking system call
      - Lose "on any system" advantage, page faults still a problem.

Kernel Threads

- Cons
  - Thread creation and destruction, and blocking and unblocking threads requires kernel entry and exit.
    - More expensive than user-level equivalent
- Pros
  - Preemptive multithreading
  - Parallelism
    - Can overlap blocking I/O with computation
    - Can take advantage of a multiprocessor

Kernel Threads

- Cons
  - Thread creation and destruction, and blocking and unblocking threads requires kernel entry and exit.
    - More expensive than user-level equivalent
- Pros
  - Preemptive multithreading
  - Parallelism
    - Can overlap blocking I/O with computation
    - Can take advantage of a multiprocessor

Context Switch

- Thread switch must be *transparent* for threads
  - When dispatched again, thread should not notice that something else was running in the meantime (except for elapsed time)

⇒OS must save all state that affects the thread

- This state is called the *thread context*
- Switching between threads consequently results in a *context switch*.

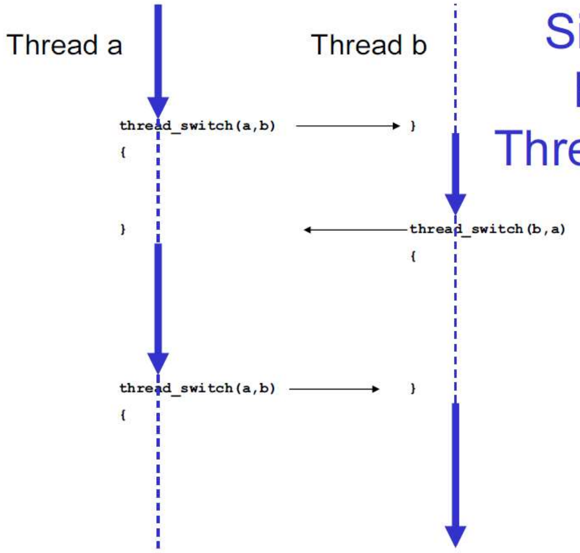
Thread Switch

- A switch between threads can happen any time the OS is invoked
  - On a system call
    - Mandatory if system call blocks or on exit();
  - On an exception
    - Mandatory if offender is killed
  - On an interrupt
    - Triggering a dispatch is the main purpose of the *timer interrupt*

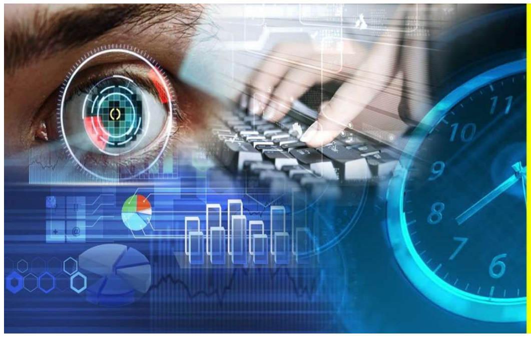
A thread switch can happen between any two instructions

Note instructions do not equal program statements

Simplified Explicit Thread Switch



SLIATE moodle HNDIT 3052 Operating Systems



HNDIT 3052 Operating Systems

Process Synchronization

Objectives Expected

- To present the concept of process synchronization.
- To present the concept of Race Condition.
- To present the concept of Critical Section Problem.
  - Peterson solution for critical section problem.
- Present the Classical problems of synchronization.
  - Bounded-Buffer Problem
  - Readers and Writers Problem
  - Dining-Philosophers Problem

Week4: Process Synchronization

Topic:

- Process Synchronization

Subtopics:

- Race Conditions
- Critical Section Problem

Introduction of Process Synchronization

- **Process Synchronization** is the coordination of execution of multiple processes in a multi-process system to ensure that they access shared resources in a controlled and predictable manner. It aims to resolve the problem of race conditions and other synchronization issues in a concurrent system.
- The **main objective** of process synchronization is to ensure that multiple processes access shared resources without interfering with each other and to prevent the possibility of inconsistent data due to concurrent access. To achieve this, various synchronization techniques such as semaphores, monitors, and critical sections are used.

Introduction of Process Synchronization . .

- In a multi-process system, synchronization is necessary to ensure data consistency and integrity, and to avoid the risk of deadlocks and other synchronization problems.
- Process synchronization is an important aspect of modern operating systems, and it plays a crucial role in ensuring the correct and efficient functioning of multi-process systems.

1. Race Condition

- When more than one process is executing the same code or accessing the same memory or any shared variable in that condition there is a possibility that the output or the value of the shared variable is wrong so for that all the processes doing the race to say that my output is correct this condition known as a race condition.
- Several processes access and process the manipulations over the same data concurrently, and then the outcome depends on the particular order in which the access takes place.
- A **race condition** is a situation that may occur inside a critical section. This happens when the result of multiple thread execution in the critical section differs according to the order in which the threads execute.
- Race conditions in critical sections can be avoided if the critical section is treated as an atomic instruction. Also, proper thread synchronization using locks or atomic variables can prevent race conditions.

On the basis of synchronization, processes are categorized as one of the following two types:

- **Independent Process**: The execution of one process does not affect the execution of other processes.
- **Cooperative Process**: A process that can affect or be affected by other processes executing in the system.

Process synchronization problem arises in the case of Cooperative processes also because resources are shared in Cooperative processes.

Example – (Race Condition)

Let’s understand one example to understand the race condition better:

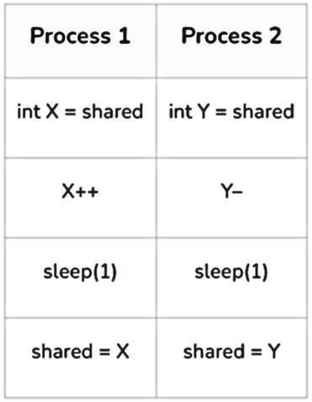
- Let’s say there are two processes P1 and P2 which share a common variable (shared=10), both processes are present in – queue and waiting for their turn to be executed.
- Suppose, Process P1 first come under execution, and the CPU store a common variable between them (shared=10) in the local variable (X=10) and increment it by 1(X=11), after then when the CPU read line sleep(1),it switches from current process P1 to process P2 present in ready-queue.
- The process P1 goes in a waiting state for 1 second.

(continue to next slide. . .)

Example – (Race Condition) . . .

- Now CPU execute the Process P2 line by line and store common variable (Shared=10) in its local variable (Y=10) and decrement Y by 1(Y=9),
- after then when CPU read sleep(1), the current process P2 goes in waiting for state and CPU remains idle for some time as there is no process in ready-queue,
- after completion of 1 second of process P1 when it comes in ready-queue, CPU takes the process P1 under execution and execute the remaining line of code (store the local variable (X=11) in common variable (shared=11)), CPU remain idle for sometime waiting for any process in ready-queue,
- after completion of 1 second of Process P2, when process P2 comes in ready-queue, CPU start executing the further remaining line of Process P2(store the local variable (Y=9) in common variable (shared=9)).
- **Initially Shared = 10**

- **Note:** We are assuming the final value of a common variable(shared) after execution of Process P1 and Process P2 is 10 (as Process P1 increment variable (shared=10) by 1 and Process P2 decrement variable (shared=11) by 1 and finally it becomes shared=10).
- But we are getting undesired value due to a lack of proper synchronization.



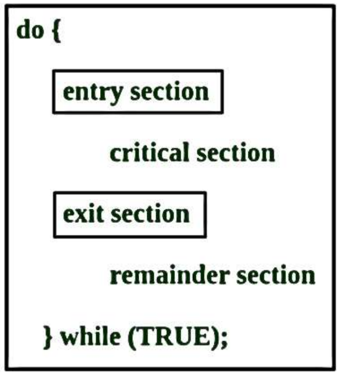
Process 1	Process 2
int X = shared	int Y = shared
X++	Y–
sleep(1)	sleep(1)
shared = X	shared = Y

Actual meaning of race-condition

- If the order of execution of the process(first P1 -> then P2) then we will get the value of common variable (shared) =9.
- If the order of execution of the process(first P2 -> then P1) then we will get the final value of common variable (shared) =11.
- Here the (value1 = 9) and (value2=10) are racing, If we execute these two processes in our computer system then sometime we will get 9 and sometime we will get 10 as the final value of a common variable(shared). This phenomenon is called race condition.

2. Critical Section Problem

- A critical section is a code segment that can be accessed by only one process at a time. The critical section contains shared variables that need to be synchronized to maintain the consistency of data variables.
- So the **critical section** problem means designing a way for cooperative processes to access shared resources without creating data inconsistencies.



```
do {
    entry section
        critical section
    exit section
        remainder section
 } while (TRUE);
```

In the entry section, the process requests for entry in the **Critical Section.**

Any solution to the critical section problem must satisfy three requirements:

- **Mutual Exclusion**: If a process is executing in its critical section, then no other process is allowed to execute in the critical section.
- **Progress**: If no process is executing in the critical section and other processes are waiting outside the critical section, then only those processes that are not executing in their remainder section can participate in deciding which will enter the critical section next, and the selection can not be postponed indefinitely.
- **Bounded Waiting**: A bound must exist 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.

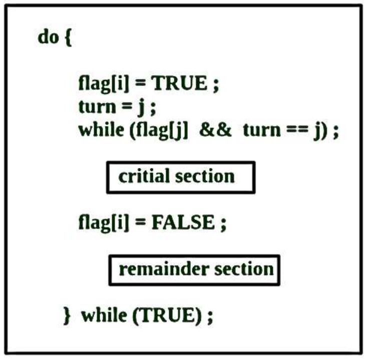
Peterson’s Solution preserves all three conditions

- Mutual Exclusion is assured as only one process can access the critical section at any time.
- Progress is also assured, as a process outside the critical section does not block other processes from entering the critical section.
- Bounded Waiting is preserved as every process gets a fair chance.

Peterson’s Solution(to the critical section problem)

Peterson’s Solution is a classical software-based solution to the critical section problem. In Peterson’s solution, we have two shared variables:

- boolean flag[i]: Initialized to FALSE, initially no one is interested in entering the critical section
- int turn: The process whose turn is to enter the critical section.



```
do {

     flag[i] = TRUE ;
     turn = j ;
     while (flag[j]  &&  turn == j) ;

          critial section

     flag[i] = FALSE ;

          remainder section

   }  while (TRUE) ;
```

Disadvantages of Peterson’s Solution

- It involves busy waiting. (In the Peterson’s solution, the code statement- “while(flag[j] && turn == j);” is responsible for this. Busy waiting is not favored because it wastes CPU cycles that could be used to perform other tasks.)
- It is limited to 2 processes.
- Peterson’s solution cannot be used in modern CPU architectures.

Classical Problems of Synchronization

- Classical problems used to test newly-proposed synchronization schemes
  - Bounded-Buffer Problem
  - Readers and Writers Problem
  - Dining-Philosophers Problem

Bounded-Buffer Problem

- ***n*** buffers, each can hold one item
- Semaphore **`mutex`** initialized to the value 1
- Semaphore **`full`** initialized to the value 0
- Semaphore **`empty`** initialized to the value n

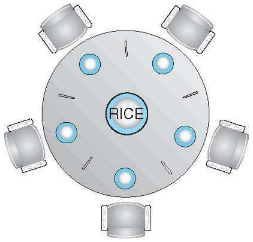
Readers-Writers Problem

- A data set is shared among a number of concurrent processes
  - Readers – only read the data set; they do ***not*** perform any updates
  - Writers – can both read and write
- Problem – allow multiple readers to read at the same time
  - Only one single writer can access the shared data at the same time
- Several variations of how readers and writers are considered – all involve some form of priorities
- Shared Data
  - Data set
  - Semaphore **`rw_mutex`** initialized to 1
  - Semaphore **`mutex`** initialized to 1
  - Integer **`read_count`** initialized to 0

Readers-Writers Problem Variations

- ***First*** variation – no reader kept waiting unless writer has permission to use shared object
- ***Second*** variation – once writer is ready, it performs the write ASAP
- Both may have starvation leading to even more variations
- Problem is solved on some systems by kernel providing reader-writer locks

Dining-Philosophers Problem



- Philosophers spend their lives alternating thinking and eating
- Don’t interact with their neighbors, occasionally try to pick up 2 chopsticks (one at a time) to eat from bowl
  - Need both to eat, then release both when done
- In the case of 5 philosophers
  - Shared data
    - Bowl of rice (data set)
    - Semaphore chopstick [5] initialized to 1

Solution to Dining Philosophers (Cont.)

- Each philosopher *i* invokes the operations **`pickup()`** and **`putdown()`** in the following sequence:

```
DiningPhilosophers.pickup(i);

    EAT

DiningPhilosophers.putdown(i);
```

- No deadlock, but starvation is possible

Dining-Philosophers Problem Algorithm (Cont.)

- Deadlock handling
  - Allow at most 4 philosophers to be sitting simultaneously at the table.
  - Allow a philosopher to pick up the forks only if both are available (picking must be done in a critical section.
  - Use an asymmetric solution -- an odd-numbered philosopher picks up first the left chopstick and then the right chopstick. Even-numbered philosopher picks up first the right chopstick and then the left chopstick.

Advantages & Disadvantages of Process Synchronization

Advantages of Process Synchronization

- Ensures data consistency and integrity
- Avoids race conditions
- Prevents inconsistent data due to concurrent access
- Supports efficient and effective use of shared resources

Disadvantages of Process Synchronization

- Adds overhead to the system
- This can lead to performance degradation
- Increases the complexity of the system
- Can cause deadlocks if not implemented properly.

Problem and Answers

1. What is the main objective of process synchronization in a multi-process system?

   Answer: A multi-process system's process synchronization goal is to govern and predict shared resource access. It prevents race situations and other synchronization concerns to maintain data integrity and avoid deadlocks.

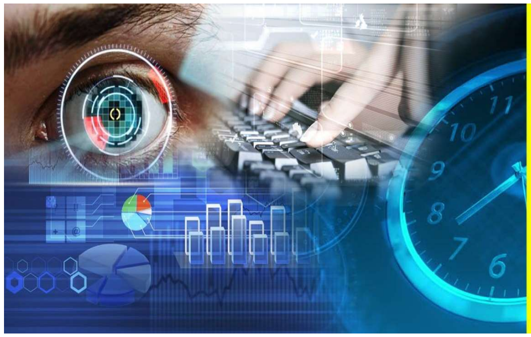
2. What are the key requirements that any solution to the critical section problem must satisfy?

   Answer:

   The crucial section problem solution must meet three criteria:

   - **Mutual Exclusion:** Only one process can run in its critical section.
   - **Progress:** Only processes not in their remainder sections can select the next process to enter the critical section if no process is in it and others are waiting.
   - **Bounded Waiting:** The number of times a process can enter its crucial phase after making a request must be limited before it is granted.

Next→ Process Synchronization



HNDIT 3052 Operating Systems

Process Scheduling

Chapter 6: CPU Scheduling

- Basic Concepts and Introduction
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multiple-Processor Scheduling
- Real-Time CPU Scheduling

Topic: Process Scheduling

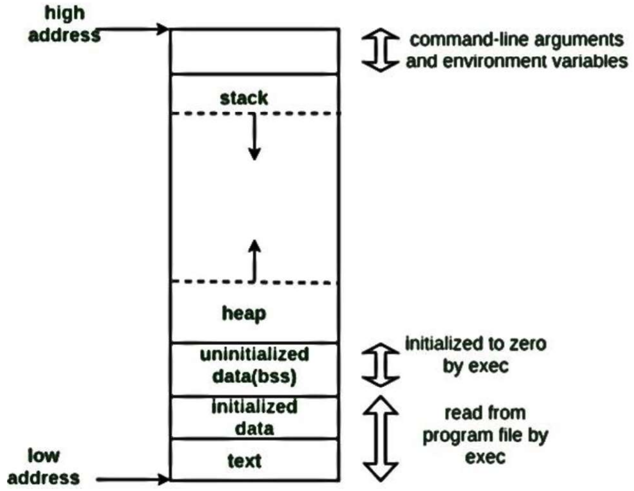
- Subtopics:
  1. Introduction to Scheduling
  2. Scheduling Algorithms

Objectives

- To introduce CPU scheduling, which is the basis for multiprogrammed operating systems
- To describe various CPU-scheduling algorithms
- To discuss evaluation criteria for selecting a CPU-scheduling algorithm for a particular system
- To examine the scheduling algorithms of several operating systems

What is a process?

- In computing, a process is **the instance of a computer program that is being executed by one or many threads**. It contains the program code and its activity. Depending on the operating system (OS), a process may be made up of multiple threads of execution that execute instructions concurrently.


high address
command-line arguments and environment variables
stack
heap
uninitialized data(bss)
initialized to zero by exec
initialized data
read from program file by exec
text
low address

How is process memory used for efficient operation?

The process memory is divided into four sections for efficient operation:

- The **text category** is composed of integrated program code, which is read from fixed storage when the program is launched.
- The **data class** is made up of global and static variables, distributed and executed before the main action.
- Heap is used for flexible, or dynamic memory allocation and is managed by calls to new, delete, malloc, free, etc.
- The stack is used for local variables. The space in the stack is reserved for local variables when it is announced.

States of a Process in Operating Systems

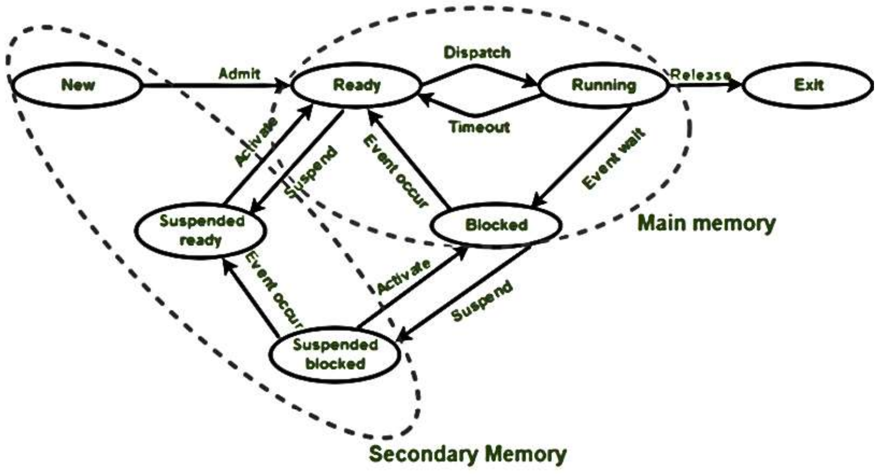
- A process has several stages that it passes through from beginning to end. There must be a minimum of five states. Even though during execution, the process could be in one of these states, the names of the states are not standardized. Each process goes through several stages throughout its life cycle.

Process States in Operating System

The states of a process are as follows:

- **New (Create):** In this step, the process is about to be created but not yet created. It is the program that is present in secondary memory that will be picked up by OS to create the process.
- **Ready:** New -> Ready to run. After the creation of a process, the process enters the ready state i.e. the process is loaded into the main memory. The process here is ready to run and is waiting to get the CPU time for its execution. Processes that are ready for execution by the CPU are maintained in a queue called ready queue for ready processes.
- **Run:** The process is chosen from the ready queue by the CPU for execution and the instructions within the process are executed by any one of the available CPU cores.
- **Blocked or Wait:** Whenever the process requests access to I/O or needs input from the user or needs access to a critical region(the lock for which is already acquired) it enters the blocked or waits for the state. The process continues to wait in the main memory and does not require CPU. Once the I/O operation is completed the process goes to the ready state.
- **Terminated or Completed:** Process is killed as well as PCB is deleted. The resources allocated to the process will be released or deallocated.
- **Suspend Ready:** Process that was initially in the ready state but was swapped out of main memory(refer to Virtual Memory topic) and placed onto external storage by the scheduler is said to be in suspend ready state. The process will transition back to a ready state whenever the process is again brought onto the main memory.
- **Suspend wait or suspend blocked:** Similar to suspend ready but uses the process which was performing I/O operation and lack of main memory caused them to move to secondary memory. When work is finished it may go to suspend ready.

- **How does a process move between different states in an operating system?**
  A process can move between different states in an operating system based on its execution status and resource availability. Here are some examples of how a process can move between different states:



CPU and I/O Bound Processes: If the process is intensive in terms of CPU operations, then it is called CPU bound process. Similarly, If the process is intensive in terms of I/O operations then it is called I/O bound process.

How does a process move between different states in an operating system?

- **New to ready:** When a process is created, it is in a new state. It moves to the ready state when the operating system has allocated resources to it and it is ready to be executed.
- **Ready to running:** When the CPU becomes available, the operating system selects a process from the ready queue depending on various scheduling algorithms and moves it to the running state.
- **Running to blocked:** When a process needs to wait for an event to occur (I/O operation or system call), it moves to the blocked state. For example, if a process needs to wait for user input, it moves to the blocked state until the user provides the input.
- **Running to ready:** When a running process is preempted by the operating system, it moves to the ready state. For example, if a higher-priority process becomes ready, the operating system may preempt the running process and move it to the ready state.
- **Blocked to ready:** When the event a blocked process was waiting for occurs, the process moves to the ready state. For example, if a process was waiting for user input and the input is provided, it moves to the ready state.
- **Running to terminated:** When a process completes its execution or is terminated by the operating system, it moves to the terminated state.

What is Process Scheduling?

- Process Scheduling is the process of the process manager handling the removal of an active process from the CPU and selecting another process based on a specific strategy.
- Process Scheduling is an integral part of Multi-programming applications. Such operating systems allow more than one process to be loaded into usable memory at a time and the loaded shared CPU process uses repetition time.
- There are three **types of process schedulers**:
  1. Long term or **Job Scheduler**
  2. Short term or **CPU Scheduler**
  3. Medium-term Scheduler

What is the need for CPU scheduling algorithm?

- **CPU scheduling** is the process of deciding which process will own the CPU to use while another process is suspended. The main function of the CPU scheduling is to ensure that whenever the CPU remains idle, the OS has at least selected one of the processes available in the ready-to-use line.
- In **Multiprogramming**, if the long-term scheduler selects multiple I / O binding processes then most of the time, the CPU remains an idle. The function of an effective program is to improve resource utilization.
- If most operating systems change their status from performance to waiting then there may always be a chance of failure in the system. So in order to minimize this excess, the OS needs to schedule tasks in order to make full use of the CPU and avoid the possibility of deadlock.

Why do we need to schedule processes?

- **Scheduling** is important in many different computer environments. One of the most important areas is scheduling which programs will work on the CPU. This task is handled by the Operating System (OS) of the computer and there are many different ways in which we can choose to configure programs.
- **Process Scheduling** allows the OS to allocate CPU time for each process. Another important reason to use a process scheduling system is that it keeps the CPU busy at all times. This allows you to get less response time for programs.

What are the different terminologies to take care of in any CPU Scheduling algorithm?

- **Arrival Time:** Time at which the process arrives in the ready queue.
- **Completion Time:** Time at which process completes its execution.
- **Burst Time:** Time required by a process for CPU execution.
- **Turn Around Time:** Time Difference between completion time and arrival time.

Turn Around Time = Completion Time – Arrival Time

- **Waiting Time(W.T):** Time Difference between turn around time and burst time.

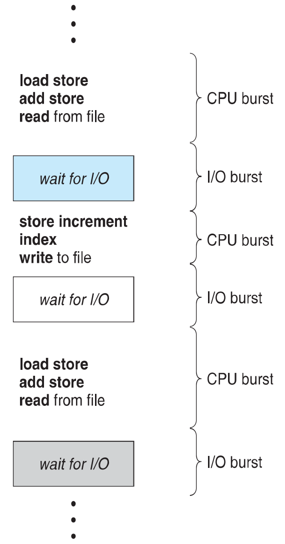
Waiting Time = Turn Around Time – Burst Time

What are the different types of CPU Scheduling Algorithms?

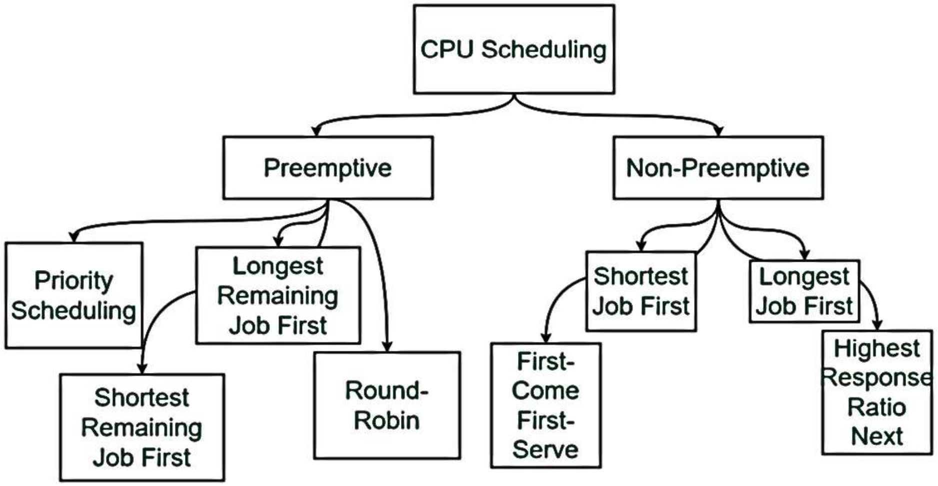
- There are mainly two types of scheduling methods:
- **Preemptive Scheduling:** Preemptive scheduling is used when a process switches from running state to ready state or from the waiting state to the ready state.
- **Non-Preemptive Scheduling:** Non-Preemptive scheduling is used when a process terminates , or when a process switches from running state to waiting state.

Basic Concepts

- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a **cycle** of CPU execution and I/O wait
- **CPU burst** followed by **I/O burst**
- CPU burst distribution is of main concern



Classification of different types of CPU Scheduling Algorithms?



Different types of CPU Scheduling Algorithms

About CPU scheduling algorithms in operating systems:

1. **First Come First Serve(FCFS).**
2. **Shortest Job First(SJF).**
3. **Priority Scheduling.**
4. **Round robin.**
5. **Shortest Remaining Time First.**
6. **etc..**

In this section we discuss **only** No.1 and No.5

1. First Come First Serve(FCFS).

FCFS considered to be the simplest of all operating system scheduling algorithms. First come first serve scheduling algorithm states that the process that requests the CPU first is allocated the CPU first and is implemented by using FIFO queue.

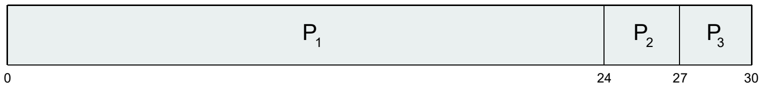
Characteristics of FCFS:

- FCFS supports non-preemptive and preemptive CPU scheduling algorithms.
- Tasks are always executed on a First-come, First-serve concept.
- FCFS is easy to implement and use.
- This algorithm is not much efficient in performance, and the wait time is quite high.

1. Example-01: First- Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:



- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

Advantages and Disadvantages of FCFS:

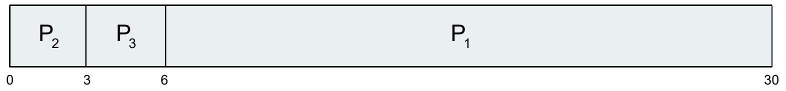
- **Advantages of FCFS:**
  - Easy to implement.
  - First come, first serve method.
- **Disadvantages of FCFS:**
  - FCFS suffers from **Convoy effect**.
  - The average waiting time is much higher than the other algorithms.
  - FCFS is very simple and easy to implement and hence not much efficient.

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order:

P_2 , P_3 , P_1

- The Gantt chart for the schedule is:



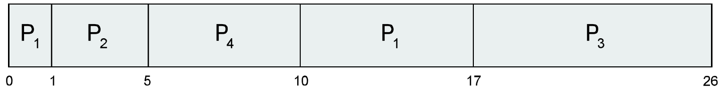
- Waiting time for P_1 = 6; P_2 = 0$_{,}$ P_3 = 3
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case
- **Convoy effect** - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes

5. Example of Shortest-remaining-time-first

- Now we add the concepts of varying arrival times and preemption to the analysis

Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- *Preemptive* SJF Gantt Chart

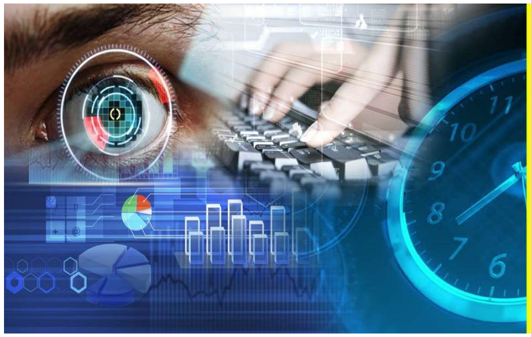


P_1 | P_2 | P_4 | P_1 | P_3

0 1 5 10 17 26

- Average waiting time = [(10-1)+(1-1)+(17-2)+5-3)]/4 = 26/4 = 6.5 msec

End of Chapter PROCESS/ CPU SCHEDULING



HNDIT 3052 Operating Systems

Deadlocks

Topic:

- Deadlocks

Subtopics:

1. Introduction to Deadlocks
2. Deadlock Detection and Recovery
3. Deadlock Avoidance
4. Deadlock Prevention

- **Definition.** A deadlock is a condition that may happen in a system composed of multiple processes that can access shared resources. A deadlock is said to occur when two or more processes are waiting for each other to release a resource. None of the processes can make any progress.

Introduction of Deadlock in Operating System

- A deadlock is **a situation in which two computer programs sharing the same resource are effectively preventing each other from accessing the resource**, resulting in both programs ceasing to function.
- The earliest computer operating systems ran only one program at a time. All of the resources of the system were available to this one program.
- Later, operating systems ran multiple programs at once, interleaving them. Programs were required to specify in advance what resources they needed so that they could avoid conflicts with other programs running at the same time.
- Eventually some operating systems offered dynamic allocation of resources. Programs could request further allocations of resources after they had begun running. This **led to the problem of the deadlock.**

Here is the simplest example:

Program 1 requests resource A and receives it.

Program 2 requests resource B and receives it.

Program 1 requests resource B and is queued up, pending the release of B.

Program 2 requests resource A and is queued up, pending the release of A.

- Now neither program can proceed until the other program releases a resource. The operating system cannot know what action to take. At this point the only alternative is to abort (stop) one of the programs.
- Learning to deal with deadlocks had a major impact on the development of operating systems and the structure of **databases**. Data was structured and the order of requests was constrained in order to avoid creating deadlocks.

A Deadlock

- *A* ***deadlock*** is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

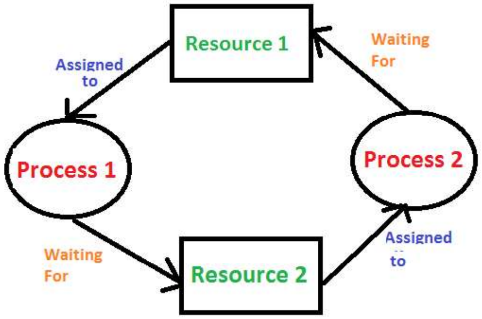
How does it uses resources in Operating Systems?

- A process in operating system uses resources in the following way.

1. Requests a resource
2. Use the resource
3. Releases the resource

Deadlock Example:

- Consider an example when **two trains are coming toward each other on the same track** and there is only one track, none of the trains can move once they are in front of each other.
- A similar situation occurs in operating systems when **there are two or more processes that hold some resources and wait for resources held by other(s)**.
- For example, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.



Examples Of Deadlock

1. The system has 2 tape drives. P1 and P2 each hold one tape drive and each needs another one.
2. Semaphores A and B, initialized to 1, P0, and P1 are in deadlock as follows:
   - P0 executes wait(A) and preempts.
   - P1 executes wait(B).
   - Now P0 and P1 enter in deadlock.

P0	P1
wait(A);	wait(B)
wait(B);	wait(A)

3. Assume the space is available for allocation of 200K bytes, and the following sequence of events occurs.

P0	P1
Request 80KB;	Request 70KB;
Request 60KB;	Request 80KB;

Deadlock occurs if both processes progress to their second request.

Methods for handling deadlock

- There are three ways to handle deadlock .
  1. Deadlock prevention or avoidance:
  2. Deadlock detection and recovery:
  3. Deadlock ignorance:

Deadlock can arise if the following four conditions hold simultaneously (Necessary Conditions):

1. ***Mutual Exclusion:*** Two or more resources are non-shareable (Only one process can use at a time).
2. ***Hold and Wait:*** A process is holding at least one resource and waiting for resources.
3. ***No Preemption:*** A resource cannot be taken from a process unless the process releases the resource.
4. ***Circular Wait:*** A set of processes waiting for each other in circular form.

1) Deadlock prevention or avoidance:

1.1 Prevention:

- The idea is to not let the system into a deadlock state. This system will make sure that above mentioned four conditions will not arise. These techniques are very costly so we use this in cases where our priority is making a system deadlock-free. One can zoom into each category individually, Prevention is done by negating one of the above-mentioned necessary conditions for deadlock.
- Prevention can be done in four different ways:

  1.1.1. Eliminate **mutual exclusion**

  1.1.2. Solve **hold and Wait**

  1.1.3. Allow **preemption**

  1.1.4. **Circular wait** Solution

Deadlock Prevention

Restrain the ways request can be made

- **Mutual Exclusion** – not required for sharable resources (e.g., read-only files); must hold for non-sharable resources
- **Hold and Wait** – must guarantee that whenever a process requests a resource, it does not hold any other resources
  - Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none allocated to it.
  - Low resource utilization; starvation possible

Deadlock Prevention (Cont.)

- **No Preemption** –
  - 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
  - Preempted 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** – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration

1.2 Avoidance:

- Avoidance is kind of ***futuristic***. By using the strategy of “Avoidance”, we have to make an assumption.
- We need to ensure that all information about resources that the process will need is known to us before the execution of the process.
- We use **Banker’s algorithm**

  (Which is in turn a gift from Dijkstra) to avoid deadlock.
- In prevention and avoidance, we get the correctness of data but performance decreases.

2) Deadlock detection and recovery:

- If Deadlock prevention or avoidance is not applied to the software then we can handle this by deadlock detection and recovery. which consist of two phases:

1. In the first phase, we examine the state of the process and check whether there is a deadlock or not in the system.
2. If found deadlock in the first phase then we apply the algorithm for recovery of the deadlock.

- In Deadlock detection and recovery, we get the correctness of data **but** performance decreases.

3) Deadlock ignorance:

- If a deadlock is very rare, then let it happen and reboot the system. This is the approach that both Windows and UNIX take. we use the ostrich algorithm for deadlock ignorance.
- In Deadlock, ignorance performance is better than the above two methods but the correctness of data.

Safe State: A safe state can be defined as a state in which there is no deadlock. It is achievable if:

1. If a process needs an unavailable resource, it may wait until the same has been released by a process to which it has already been allocated. if such a sequence does not exist, it is an unsafe state.
2. All the requested resources are allocated to the process.

Deadlocks

- System Model
- Deadlock Characterization
- Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- Recovery from Deadlock

Deadlocks

Section Objectives

- To develop a description of deadlocks, which prevent sets of concurrent processes from completing their tasks.
- To present a number of different methods for preventing or avoiding deadlocks in a computer system.

System Model

- System consists of resources
- Resource types $R_1, R_2, \ldots, R_m$

  CPU cycles, memory space, I/O devices
- Each resource type R_i has W_i instances.
- Each process utilizes a resource as follows:
  - **request**
  - **use**
  - **release**

Deadlock with Mutex Locks

- Deadlocks can occur via system calls, locking, etc.

Deadlock Characterization

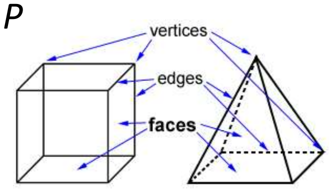
Deadlock can arise if four conditions hold simultaneously.

- **Mutual exclusion**: only one process at a time can use a resource
- **Hold and wait**: a process holding at least one resource is waiting to acquire additional resources held by other processes
- **No preemption**: a resource can be released only voluntarily by the process holding it, after that process has completed its task
- **Circular wait**: there exists a set $\{P_0, P_1, \ldots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by $P_2, \ldots, P_{n-1}$ is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.

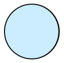
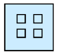
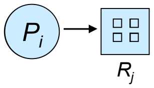
Resource-Allocation Graph

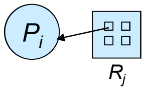
A set of vertices *V* and a set of edges *E*.

- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system
- **request edge** – directed edge $P_i \rightarrow R_j$
- **assignment edge** – directed edge $R_j \rightarrow P$

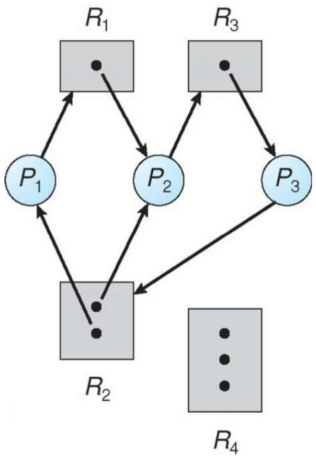


Resource-Allocation Graph (Cont.)

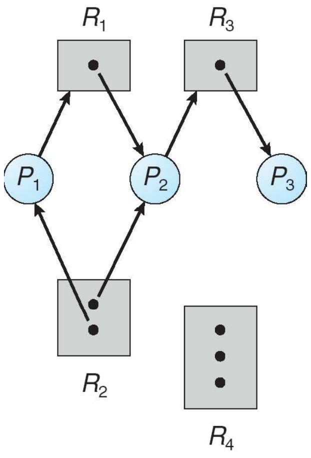
- Process 
- Resource Type with 4 instances 
- P_i requests instance of R_j 

- P_i is holding an instance of R_j 

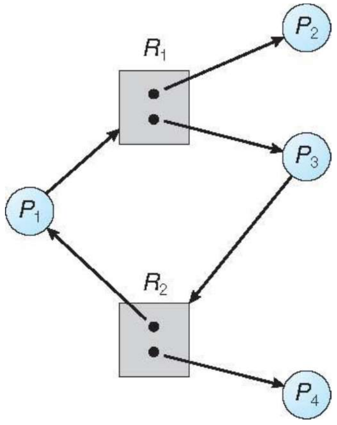
Resource Allocation Graph With A Deadlock



Example of a Resource Allocation Graph



Graph With A Cycle But No Deadlock



Basic Facts

- If graph contains no cycles ⇒ no deadlock
- If graph contains a cycle ⇒
  - if only one instance per resource type, then deadlock
  - if several instances per resource type, possibility of deadlock

Data Structures for the Banker's Algorithm

Let n = number of processes, and m = number of resources types.

- **Available**: Vector of length m. If available $[j] = k$, there are k instances of resource type R_j available
- **Max**: $n \times m$ matrix. If $Max\ [i,j] = k$, then process P_i may request at most k instances of resource type R_j
- **Allocation**: $n \times m$ matrix. If Allocation$[i,j] = k$ then P_i is currently allocated k instances of R_j
- **Need**: $n \times m$ matrix. If $Need[i,j] = k$, then P_i may need k more instances of R_j to complete its task

$$Need\ [i,j] = Max[i,j] – Allocation\ [i,j]$$

Banker's Algorithm

- Multiple instances
- Each process must a priori claim maximum use
- When a process requests a resource it may have to wait
- When a process gets all its resources it must return them in a finite amount of time

Example of Banker's Algorithm

- 5 processes P_0 through P_4;

  3 resource types:

  A (10 instances), *B* (5instances), and *C* (7 instances)
- Snapshot at time T_0:

	Allocation	*Max*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Example (Cont.)

- The content of the matrix ***Need*** is defined to be ***Max – Allocation***

	Need *A B C*
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

- The system is in a safe state since the sequence < P_1, P_3, P_4, P_2, P_0> satisfies safety criteria

NEXT➔ **Memory Management**

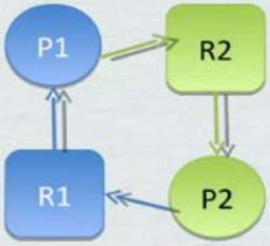
Deadlock Detection & Prevention

Deadlock solutions:

- Prevention : ✓ 1
  - Design system so that deadlock is impossible
- Avoidance : ✓ 2
  - Steer around deadlock with smart scheduling
- **Detection & recovery :** 3
  - **Check for deadlock periodically**
  - **Recover by killing a deadlocked processes and releasing its resources**
- Do nothing : ✓ 4
  - Prevention, avoidance and detection/recovery are expensive
  - Manual intervention (kill processes, reboot) if needed

Deadlock Definition :

- There exists a cycle of processes such that each process cannot proceed until the next process takes some specific action.
- Result: all processes in the cycle are stuck!



Deadlock Prevention:

#1: No mutual exclusion
- only one process at a time can use a resource.

#2: no preemption
- previously granted resources cannot forcibly taken away.

#3: No hold and wait
- When waiting for a resource, must not currently hold any resource

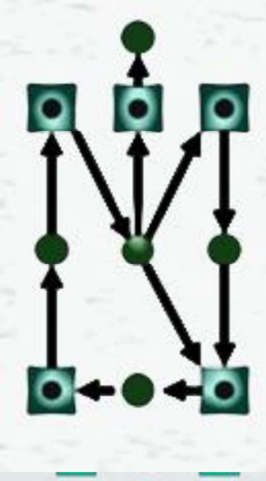
#4: circular wait condition
- Must be A circular chain of 2 or more processes
- When Waiting For Resource Must Not Currently Hold Any Resource

Deadlock Detection & Recovery

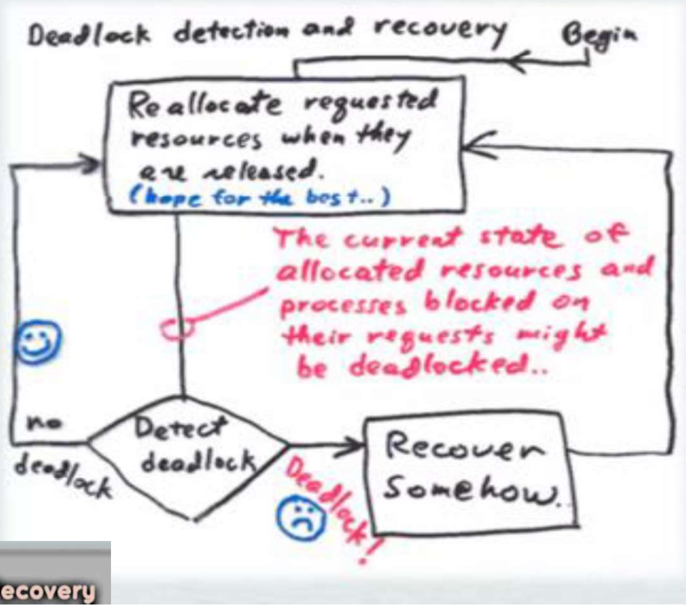
Deadlock Detection :

Check to see if a deadlock has occurred!

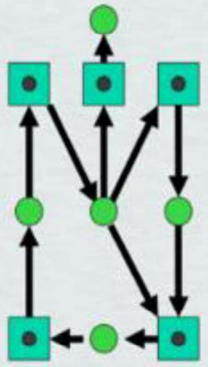
- Special case: Single resource per type
  - E.g., mutex locks (value is zero or one)
  - Check for cycles in the resource allocation graph
- General case
  - E.g., semaphores, memory pages, ...



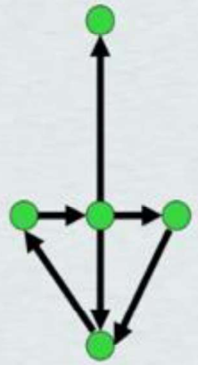
Over View OF Deadlock Detection & Recovery :


Deadlock detection and recovery
Begin
Reallocate requested resources when they are released.
(hope for the best..)
The current state of allocated resources and processes blocked on their requests might be deadlocked..
no deadlock
Detect deadlock
Deadlock!
Recover Somehow.

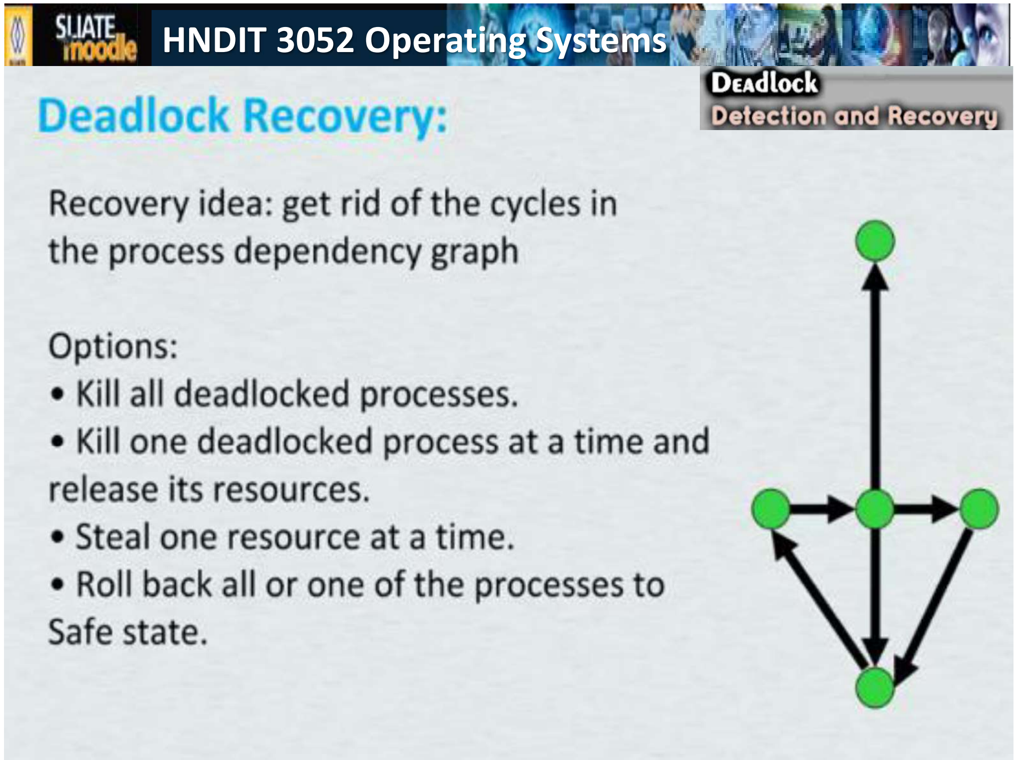
Dependencies B/W processes:



Resource allocation graph



Corresponding process dependency graph



HNDIT 3052 Operating Systems

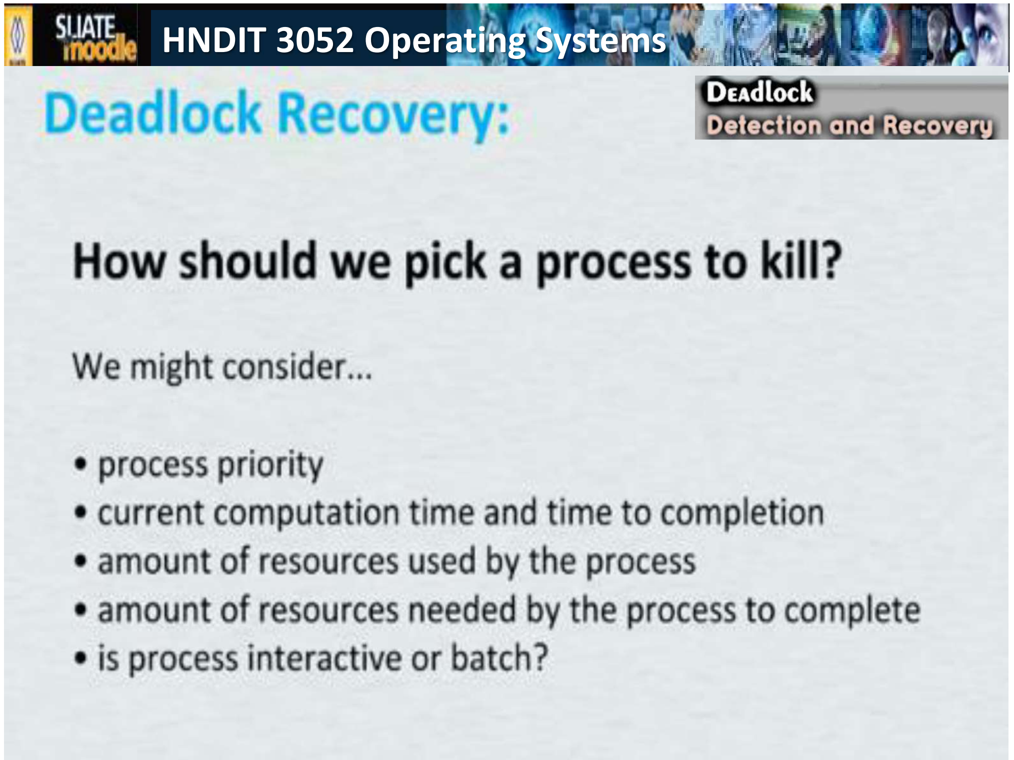
Deadlock
Detection and Recovery

Deadlock Recovery:

Recovery idea: get rid of the cycles in the process dependency graph

Options:

- Kill all deadlocked processes.
- Kill one deadlocked process at a time and release its resources.
- Steal one resource at a time.
- Roll back all or one of the processes to Safe state.



HNDIT 3052 Operating Systems

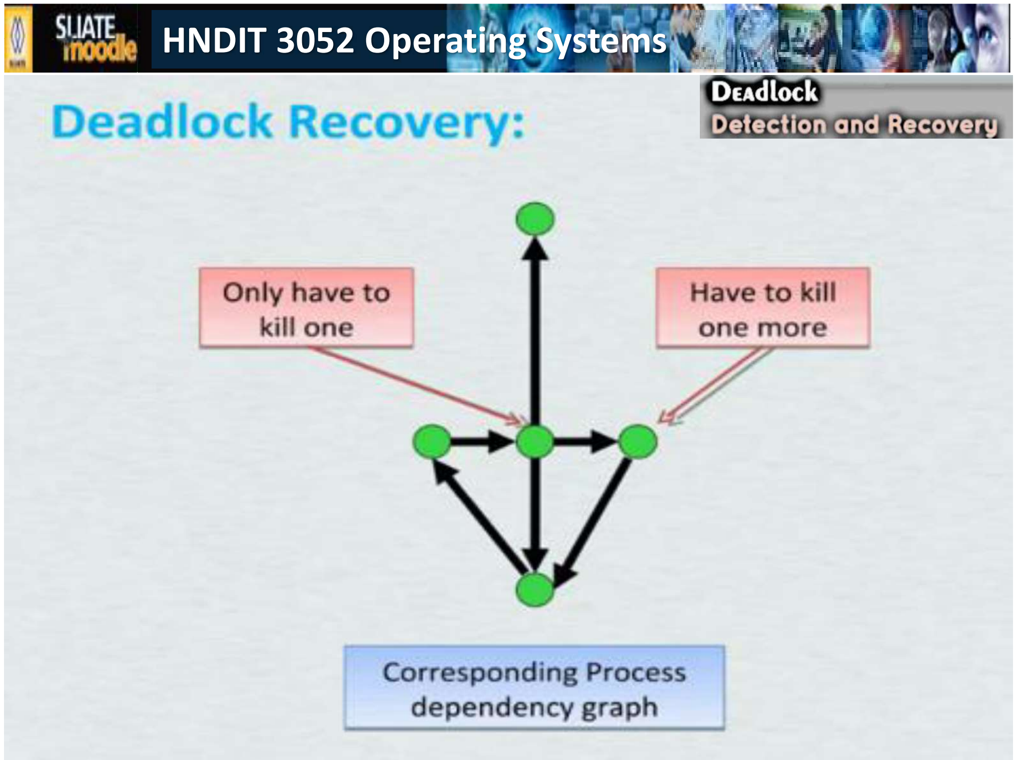
Deadlock
Detection and Recovery

Deadlock Recovery:

How should we pick a process to kill?

We might consider...

- process priority
- current computation time and time to completion
- amount of resources used by the process
- amount of resources needed by the process to complete
- is process interactive or batch?



HNDIT 3052 Operating Systems

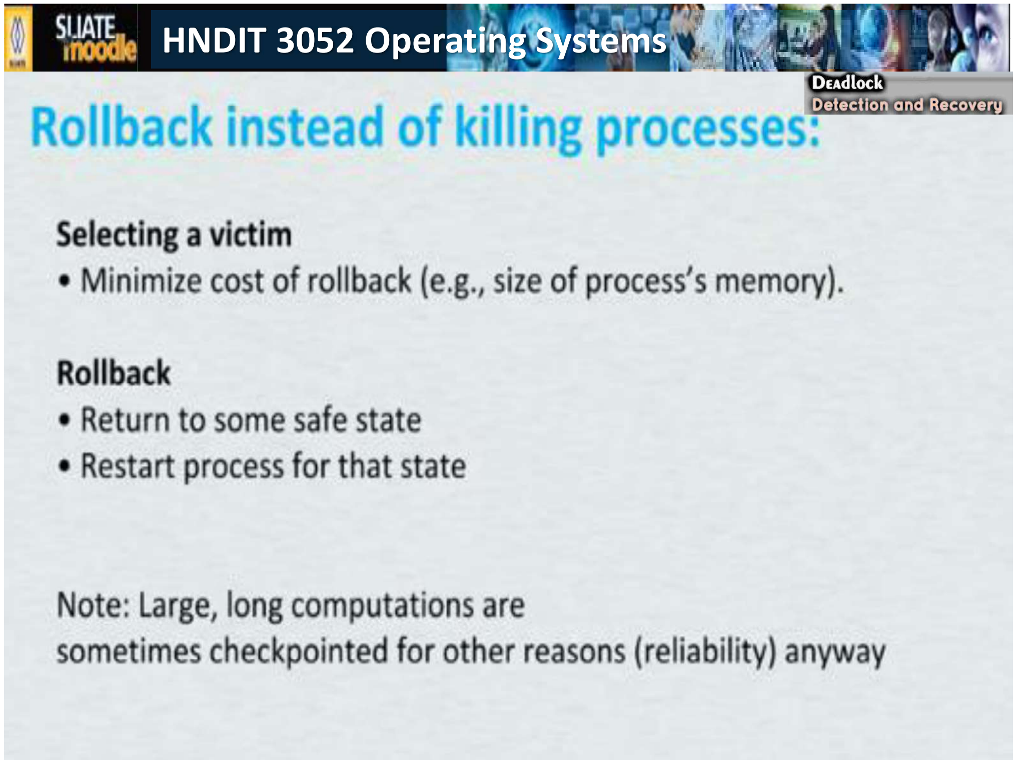
Deadlock
Detection and Recovery

Deadlock Recovery:

Only have to kill one

Have to kill one more

Corresponding Process dependency graph



HNDIT 3052 Operating Systems

Deadlock
Detection and Recovery

Rollback instead of killing processes:

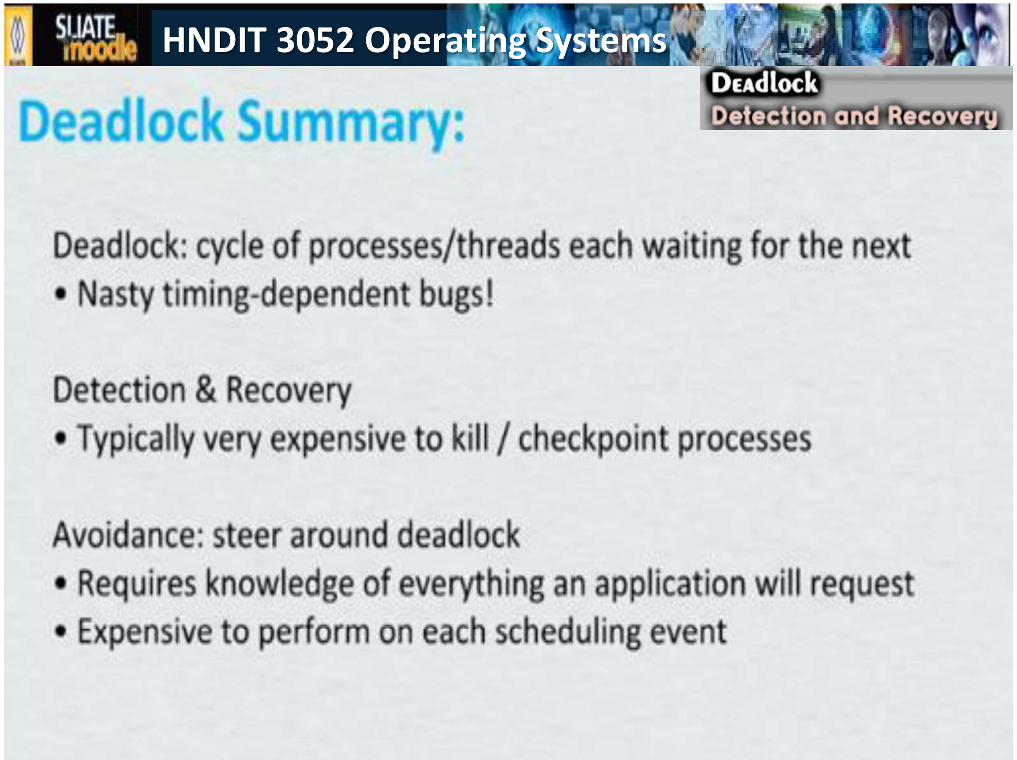
Selecting a victim

- Minimize cost of rollback (e.g., size of process's memory).

Rollback

- Return to some safe state
- Restart process for that state

Note: Large, long computations are
sometimes checkpointed for other reasons (reliability) anyway



HNDIT 3052 Operating Systems

Deadlock
Detection and Recovery

Deadlock Summary:

Deadlock: cycle of processes/threads each waiting for the next

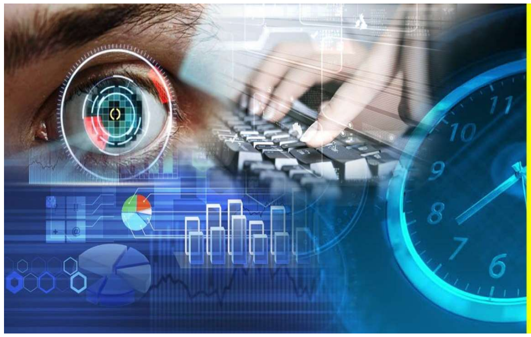
- Nasty timing-dependent bugs!

Detection & Recovery

- Typically very expensive to kill / checkpoint processes

Avoidance: steer around deadlock

- Requires knowledge of everything an application will request
- Expensive to perform on each scheduling event



HNDIT 3052 Operating Systems

Week-07: Memory Management

The Background of Main Memory

- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage CPU can access directly
- Memory unit only sees a stream of addresses + read requests, or address + data and write requests
- Register access in one CPU clock (or less)
- Main memory can take many cycles, causing a **stall**
- **Cache** sits between main memory and CPU registers
- Protection of memory required to ensure correct operation

Week-07: Memory Management

Subtopics:

1. The Notion of an Address Space
2. Swapping

Memory Management

- Memory management is the process of controlling and coordinating a computer's main memory.
- It ensures that **blocks of memory space** are:
  - properly managed and
  - Allocated
- so the
  1. operating system (OS),
  2. applications and
  3. other running processes

  have the memory they need to carry out their operations.

Memory Management in Operating System

- The term memory can be defined as a collection of data in a specific format.

  It is used to store instructions and process data.

  The memory comprises a large array or group of words or bytes, each with its own location.

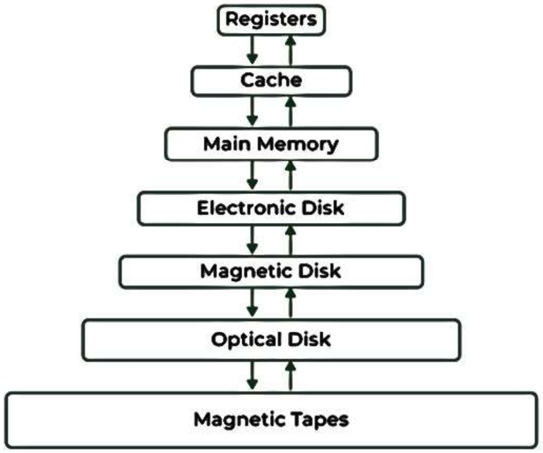
  The primary purpose of a computer system is to execute programs.

  These programs, along with the information they access, should be in the main memory during execution.

  The CPU fetches instructions from memory according to the value of the program counter.

- To achieve a degree of multiprogramming and proper utilization of memory, memory management is important.

  Many memory management methods exist, reflecting various approaches, and the effectiveness of each algorithm depends on the situation.



What is Main Memory?

- The main memory is central to the operation of a Modern Computer.
- Main Memory is a large array of words or bytes, ranging in size from hundreds of thousands to billions.
- Main memory is a repository of rapidly available information shared by the CPU and I/O devices.
- Main memory is the place where programs and information are kept when the processor is effectively utilizing them.
- Main memory is associated with the processor, so moving instructions and information into and out of the processor is extremely fast.
- Main memory is also known as RAM (Random Access Memory). This memory is volatile. RAM loses its data when a power interruption occurs.

What is Memory Management?

- In a multiprogramming computer, the Operating System resides in a part of memory, and the rest is used by multiple processes. The task of subdividing the memory among different processes is called Memory Management. Memory management is a method in the operating system to manage operations between main memory and disk during process execution. The main aim of memory management is to achieve efficient utilization of memory.

Why Memory Management is Required?

- Allocate and de-allocate memory before and after process execution.
- To keep track of used memory space by processes.
- To minimize fragmentation issues.
- To proper utilization of main memory.
- To maintain data integrity while executing of process.

Logical and Physical Address Space

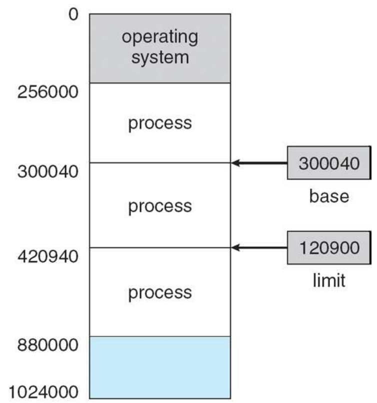
1. **Logical Address Space:** An address generated by the CPU is known as a "Logical Address". It is also known as a **Virtual address**. Logical address space can be defined as the size of the process. A logical address can be changed.
2. **Physical Address Space:** An address seen by the memory unit (i.e the one loaded into the memory address register of the memory) is commonly known as a "Physical Address". A **Physical address** is also known as a **Real address**. The set of all physical addresses corresponding to these logical addresses is known as Physical address space. A **physical address** is computed by MMU.
   - The run-time mapping from **virtual to physical** addresses is done by a hardware device Memory Management Unit(MMU). The physical address always remains constant.

Address Spaces

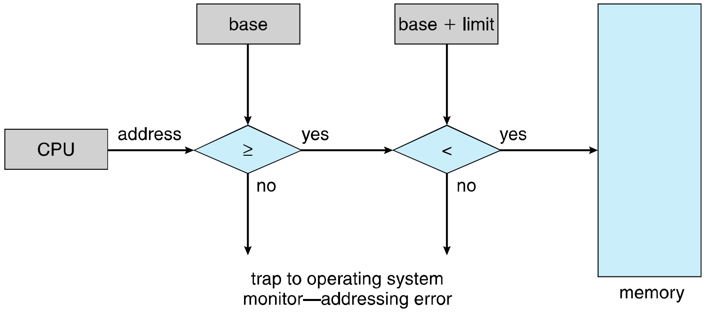
1. **Logical Address Space and**
2. **Physical Address Space**

Base and Limit Registers

- A pair of **base** and **limit registers** define the logical address space.
- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user.



Hardware Address Protection



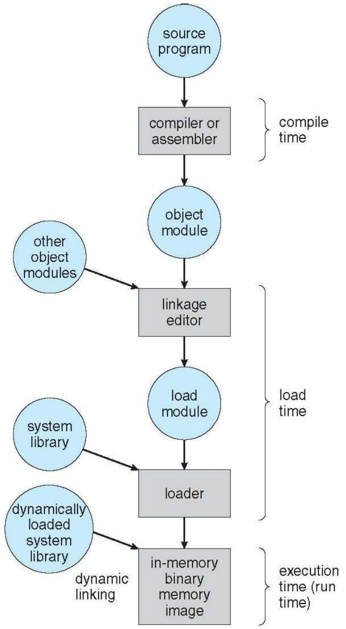
Binding of Instructions and Data to Memory

- Address binding of instructions and data to memory addresses can happen at three different stages
  - **Compile time**: If memory location known a priori, **absolute code** can be generated; must recompile code if starting location changes
  - **Load time**: Must generate **relocatable code** if memory location is not known at compile time
  - **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another
    - Need hardware support for address maps (e.g., base and limit registers)

Address Binding

- Programs on disk, ready to be brought into memory to execute form an **input queue**
  - Without support, must be loaded into address 0000
- Inconvenient to have first user process physical address always at 0000
  - How can it not be?
- Further, addresses represented in different ways at different stages of a program's life
  - Source code addresses usually symbolic
  - Compiled code addresses **bind** to relocatable addresses
    - i.e. "14 bytes from beginning of this module"
  - Linker or loader will bind relocatable addresses to absolute addresses
    - i.e. 74014
  - Each binding maps one address space to another

Multistep Processing of a User Program

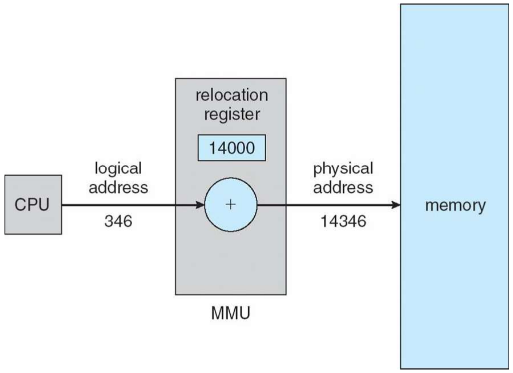


Logical vs. Physical Address Space

- The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management
  - **Logical address** – generated by the CPU; also referred to as **virtual address**
  - **Physical address** – address seen by the memory unit
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme
- **Logical address space** is the set of all logical addresses generated by a program
- **Physical address space** is the set of all physical addresses generated by a program

Dynamic relocation using a relocation register

- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- All routines kept on disk in relocatable load format
- 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



Memory-Management Unit (MMU)

- Hardware device that at run time maps virtual to physical address
- Many methods possible, covered in the rest of this chapter
- To start, consider simple scheme where the value in the relocation register is added to every address generated by a user process at the time it is sent to memory
  - Base register now called **relocation register**
  - MS-DOS on Intel 80x86 used 4 relocation registers
- The user program deals with *logical* addresses; it never sees the *real* physical addresses
  - Execution-time binding occurs when reference is made to location in memory
  - Logical address bound to physical addresses

Swapping

Swapping

- 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

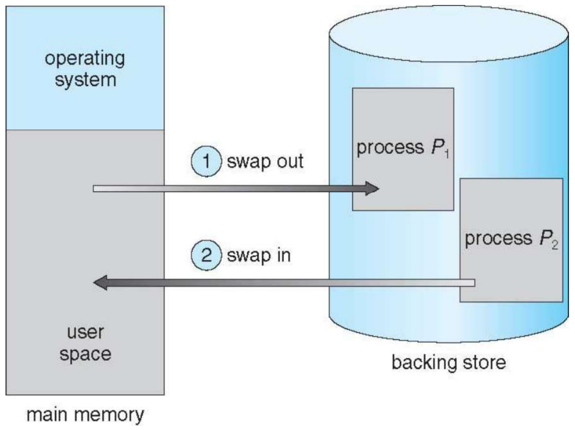
Swapping:

- When a process is executed it must have resided in memory. **Swapping** is a process of swapping a process temporarily into a secondary memory from the main memory, which is fast compared to secondary memory.
- A swapping allows more processes to be run and can be fit into memory at one time.
- The main part of swapping is transferred time and the total time is directly proportional to the amount of **memory swapped**.
- Swapping is also known as ***roll-out***, or ***roll*** because if a higher priority process arrives and wants service, the memory manager can swap out the lower priority process and then load and execute the higher priority process.
- After finishing higher priority work, the lower priority process swapped back in memory and continued to the execution process.

Swapping (…)

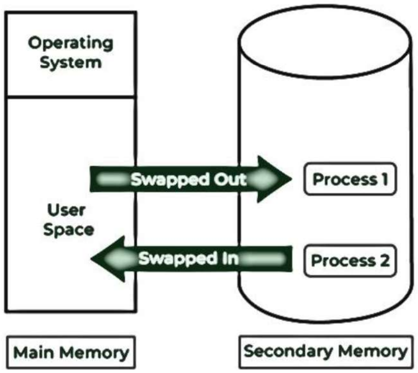
- Does the swapped out process need to swap back in to same physical addresses?
- Depends on address binding method
  - Plus consider pending I/O to / from process memory space
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
  - Swapping normally disabled
  - Started if more than threshold amount of memory allocated
  - Disabled again once memory demand reduced below threshold

Schematic View of Swapping



Swap In and **Swap** Out in OS

- The procedure by which any process gets removed from the **hard disk** and placed in the **main memory or RAM** commonly known as **Swap In.**
- On the other hand, **Swap Out** is the method of removing a process from the **main memory or RAM** and then adding it to the **Hard Disk.**



Disadvantages of Swapping

The drawbacks of the swapping technique are as follows:

1. There may occur inefficiency in the case if a resource or a variable is commonly used by those processes that are participating in the swapping process.
2. If the algorithm used for swapping is not good then the overall method can increase the number of page faults and thus decline the overall performance of processing.
3. If the computer system loses power at the time of high swapping activity then the user might lose all the information related to the program.

Advantages of Swapping

The **advantages/benefits** of the Swapping technique are as follows:

1. The swapping technique mainly helps the CPU to manage multiple processes within a single main memory.
2. This technique helps to create and use virtual memory.
3. With the help of this technique, the CPU can perform several tasks simultaneously. Thus, processes need not wait too long before their execution.
4. This technique is economical.
5. This technique can be easily applied to priority-based scheduling in order to improve its performance.

- Next→ **Virtual Memory**

Questions and Answers on address space, logical & physical address, MMU

Question1:

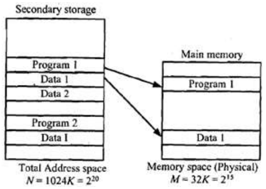
Describe about address space and memory space in Computer Science.

Answer:

An address used by a programmer will be termed as a virtual address and set of such addresses the address space. An address in main memory is known as a physical address. The set of these locations is termed as memory space. So address space is set of addresses produced by programs as they reference instructions and data, memory space comprises actual main memory locations directly addressable for processing.

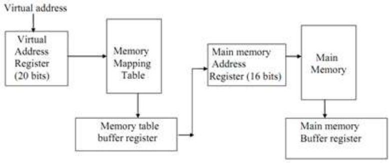
Supposes a computer which has a main-memory capacity of 64K words (K=1024). 16-bits are required to specify a physical address in memory because 64K = 2^{16}. Assume that computer has auxiliary memory for storing information equivalent to capacity of 16 main memories. Let's signify address space by N and memory space by M we then have for this illustration: N = 16 × 64 K = 1024K and M = 64K.

In a multiprogramming computer system, data and programs are transferred to and from auxiliary memory and main memory based on demands obliged by CPU. Suppose program 1 is currently being executed in CPU. Program 1 and a part of its associated data are moved from secondary memory in the main memory as displayed in Figure below. Parts of programs and data require not being in contiguous locations in memory because information is being moved in and out and empty spaces may be available in scattered locations in memory.



(Address and Memory Space in Virtual Memory)

address field of an instruction code will comprise 20 bits however physical memory addresses should be specified with just 16-bits. So, CPU will reference data and instructions with a 20 bits address though information at this address should be taken from physical memory since access to auxiliary storage for particular words will be prohibitively long. A mapping table is then required as displayed in Figure below to map a virtual address of 20 bits to a physical address of 16 bits. Mapping is a dynamic operation that means every address is translated instantly as a word is referenced by CPU.


Virtual address
Virtual Address Register (20 bits)
Memory Mapping Table
Main memory Address Register (16 bits)
Main Memory
Memory table buffer register
Main memory Buffer register

(Memory table for mapping a virtual table)

Question-02:

Describe the differences between Logical Address and Physical Address.

Answer:

S.No	Logical Address	Physical Address
1.	Users can access the **logical address** of the Program.	User can never access the **physical address** of the Program
2.	The logical address is generated by the CPU.	The physical address is located in the memory unit.
3.	The user can access the physical address with the help of a logical address.	A physical address can be accessed by a user indirectly b ut not directly.
4.	The logical address does not exist physically in the memory and thus termed as a Virtual address.	On the other hand, the physical address is a location in the memory. Thus it can be accessed physically.
5.	The set of all logical addresses that are generated by any program is referred to as **Logical Address Space.**	The set of all **physical addresses** corresponding to the **Logical addresses** is commonly known as **Physical Address Space.**
6.	This address is generated by the CPU.	It is computed by the Memory Management Unit(MMU).

Question-03:

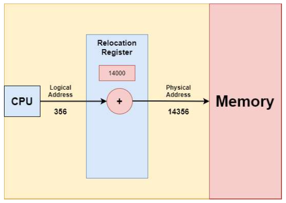
explain the memory management unit (MMU) in OS with aid of suitable example.

Answer:

Memory Management Unit(MMU) in OS

It is a hardware device that does the run-time mapping from the virtual address to the physical address. It is located within the Central Processing Unit.

Let us understand the concept of mapping with the help of a simple MMU scheme and that is a **base-register scheme**.



In the above diagram, the base register is termed the **Relocation register.** The relocation register is a special register in the CPU and is used for the mapping of logical addresses used by a program to physical addresses of the system's main memory.

The value in the relocation register is added to every address that is generated by the user process at the time when the address is sent to the memory.

MMU Example

Suppose the **base is at 14000**, then an attempt by the user to address location **0** is relocated dynamically to **14000**; thus access to location **356** is mapped to **14356.**

It is important to note that the user program never sees the real physical addresses. The Program can create a pointer to location 356 and store it in the memory and then manipulate it after that compare it with other addresses as number 356.

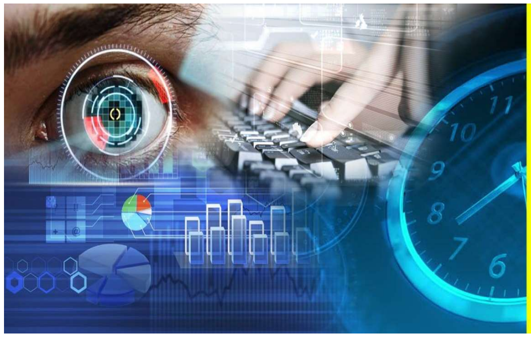
User program always deals with the logical addresses. The **Memory Mapping unit** mainly converts the logical addresses into the physical addresses. The final location of the referenced memory address is not determined **until the reference is made.**

There are two types of addresses that we have:

logical addresses (lies in the range 0 to max).

physical addresses (lies in the range R+0 to R+max for the base value R)

As we have told you above the user generates only a logical address and then thinks that the process runs in locations **0 to max.** However, these logical addresses must be mapped to physical addresses before they are used.



HNDIT 3052 Operating Systems

Virtual Memory

What is Virtual Memory

- Virtual memory uses both hardware and software to enable a computer to compensate for physical memory shortages, temporarily transferring data from random access memory (RAM) to disk storage. Mapping chunks of memory to disk files enables a computer to treat secondary memory as though it were main memory.

Virtual Memory

Topic:

1. Virtual Memory

Subtopics:

1. Paging
2. Segmentation

Virtual Memory in Operating System

- Virtual Memory is a storage allocation scheme in which **secondary memory** can be addressed as though it were part of the main memory. The addresses a program may use to reference memory are distinguished from the addresses the memory system uses to identify physical storage sites and program-generated addresses are translated automatically to the corresponding machine addresses.
- The size of virtual storage is limited by the addressing scheme of the computer system and the amount of secondary memory available not by the actual number of main storage locations.

Virtual Memory in Operating System . . .

- It is a technique that is implemented using both hardware and software. It maps memory addresses used by a program, called virtual addresses, into physical addresses in computer memory.
    1. All memory references within a process are logical addresses that are dynamically translated into **physical addresses** at run time. This means that a process can be swapped in and out of the main memory such that it occupies different places in the main memory at different times during the course of execution.
    2. A process may be broken into a number of pieces and these pieces need not be continuously located in the **main memory** during execution. The combination of dynamic run-time address translation and the use of a page or segment table permits this.

Advantages of Virtual Memory

1. **More processes may be maintained in the main memory:** Because we are going to load only some of the pages of any particular process, there is room for more processes. This leads to more efficient utilization of the processor because it is more likely that at least one of the more numerous processes will be in the ready state at any particular time.
2. **A process may be larger than all of the main memory:** One of the most fundamental restrictions in programming is lifted. A process larger than the main memory can be executed because of demand paging. The OS itself loads pages of a process in the main memory as required.
3. It allows greater multiprogramming levels by using less of the available (primary) memory for each process.
4. It has twice the capacity for addresses as main memory.
5. It makes it possible to run more applications at once.

- If these characteristics are present then, it is not necessary that all the pages or segments are present in the main memory during execution. This means that the required pages need to be loaded into memory whenever required.
- Virtual memory is implemented using:
    1. Demand **Paging** or
    2. Demand **Segmentation**.

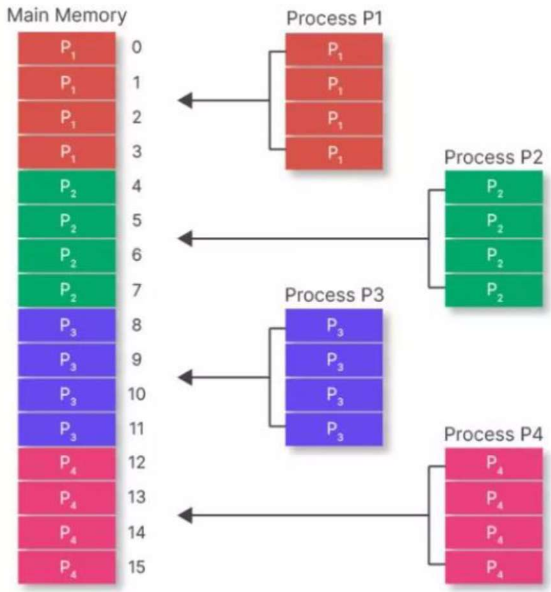
Advantages of Virtual Memory. . .

6. Users are spared from having to add memory modules when **RAM** space runs out, and applications are liberated from shared memory management.
7. When only a portion of a program is required for execution, speed has increased.
8. Memory isolation has increased security.
9. It makes it possible for several larger applications to run at once.
10. Memory allocation is comparatively cheap.
11. It doesn’t require outside fragmentation.
12. It is efficient to manage logical partition workloads using the CPU.
13. Automatic data movement is possible.

Disadvantages of Virtual Memory

1. It can slow down the system performance, as data needs to be constantly transferred between the physical memory and the hard disk.
2. It can increase the risk of data loss or corruption, as data can be lost if the hard disk fails or if there is a power outage while data is being transferred to or from the hard disk.
3. It can increase the complexity of the memory management system, as the operating system needs to manage both physical and virtual memory.

Paging:



Paging in Operating System

The Paging in OS

- In an operating system, memory management refers to dividing the memory among the various processes. The fundamental goal of memory management is to make optimal use of memory by minimizing internal and external fragmentation. One such algorithm for memory management techniques is paging.
- Operating systems utilize paging as a memory management strategy to manage memory and assign memory to programs, so it eliminates the requirement for contiguous physical memory allocation. Paging is the procedure of transferring operations in the form of pages from the secondary storage into the primary memory. Memory is split into fixed-size units called pages in paging, and processes are given memory allotments based on these pages. Paging is a logical idea that is utilized to provide quicker access to data.

What is Paging?

- **Paging** is a static memory allocation method that allows a process's physical address space to be of a non-contiguous type. It's a memory management scheme or storage mechanism that lets the operating system fetch processes from secondary memory in the form of pages and place them in the main memory. The paging hardware and operating system are integrated to implement the paging process.
- A page is a logical memory unit in a program. Logical memory is organized into equal-sized pages or equal-sized blocks. A frame is a type of physical memory unit. In the concept of paging, physical memory (main memory) is organized into frames, which are equally sized memory blocks. The memory size of a new process is determined when it arrives. If a process has n pages in local memory, there must be n frames available in the system.

Characteristics of Paging in OS

- Paging in OS has five key characteristics:
- External fragmentation is not present.
- By procedure, any frame may be employed.
- Only on the last page of a process, **internal fragmentation** may occur.
- A process's physical memory is no longer contiguous.
- A process' logical memory is still contiguous.

What is Paging? ...

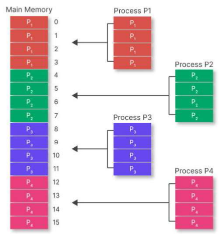
To get maximum utilization of the main memory and minimize external fragmentation, the size of a frame should be the same as the size of a page. Paging is mostly used to store non-contiguous portions of a single process.

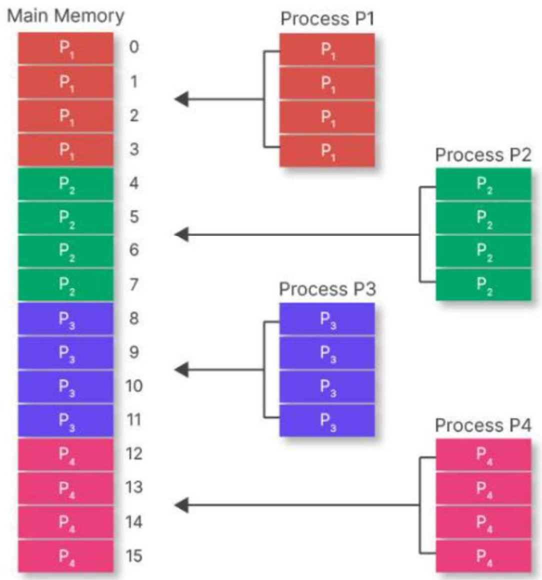
To make the paging method easier, the operating system decides:

- The program's total page count.
- Finds a sufficient number of empty page frames to assist.
- All pages must be contiguous to be loaded into memory.

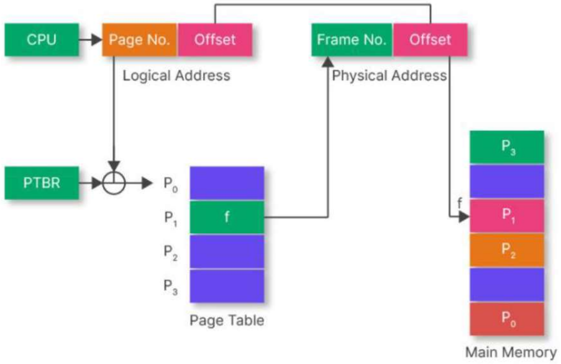
Example of Paging

- Let's say the main memory size is 64B and the frame size is 4B then, the number of frames would be 64/4 = 16.
- There are 4 processes.
- The size of each process is 16B and the page size is also 4B then, the number of pages in each process = 16/4 = 4.
- These pages may be stored in the main memory frames in a non-contiguous form, depending on their availability.





- The **required physical address** is formed by multiplying the frame number by the page offset. The **frame number** identifies the frame in which the required page is stored. **Page Offset** provides the precise word from that page that must be read.



Paging in OS: Page Table

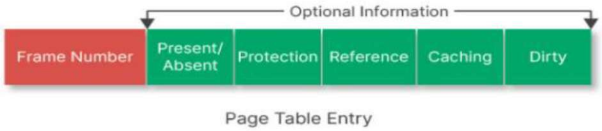
- Paging in the operating system, the logical and physical memory addresses are separated. As a result, an address translation mechanism is required to convert the logical address into a physical address. The physical address is the actual address of the frame where each page will be placed, whereas the logical address space is the address created by the CPU for each page.

The CPU generates a logical address that is made up of two parts-

- **Page Number (p):** Determines which page of the process the CPU wishes to read the data from.
- **Page Offset (d):** Defines which word on the page the CPU wants to read.

When the CPU generates a page number, the page table displays the relevant frame number (frame base address) for each page in the main memory.

PTE(Page Table Entry):

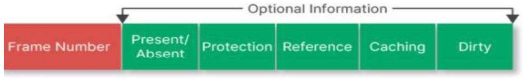


Every new process creates a separate page table (stored in physical memory). A page table entry contains a variety of page-related information. The information contained in the page table item differs from one operating system to another. **PTE(Page Table Entry)** has the following information:

1. **Frame Number:** It is the most crucial piece of information in a page table entry. The frame number identifies the memory frames in the paging in which the page is stored. The size of the frame number is determined by the number of frames in the main memory.
   1. The number of bits for frame = Size of Physical memory/Frame size.
2. **Present/Absent Bit:** This is also referred to as the valid/invalid bit. This bit indicates whether or not the page is in the main memory space. This bit is set to 0 if the page is not available in the main memory; otherwise, it is set to 1.

PTE(Page Table Entry)..

3. **Protection bit:** This bit is also known as the 'Read / Write bit.' This bit is about page security. It determines whether or not the user has permission to read and write to the page. This bit is set to 0 if only read operations are allowed and no writing operations are allowed. If both read and write operations are permitted, this bit will be set to 1.
4. **Reference bit:** The reference bit indicates whether or not the page was referred to in the previous clock cycle. If the page has recently been referenced, this bit is set to 1, otherwise, it is set to 0.
5. **Caching Enabled/Disabled:** The reference bit indicates whether or not the page was referred to in the previous clock cycle. If the page has recently been referenced, this bit is set to 1, otherwise, it is set to 0.
6. **Dirty bit:** This bit is also known as the "Modified bit." This bit indicates whether or not the page has been changed. This bit is set to 1 if the page has been updated; otherwise, it is set to 0.



Page Table Entry

Paging:

- Paging is a memory management scheme that eliminates the need for a **contiguous allocation** of physical memory.
- The process of retrieving processes in the form of pages from the secondary storage into the main memory is known as paging.
- The basic purpose of paging is to separate each procedure into pages.
- Additionally, frames will be used to split the main memory.
- This scheme permits the physical address space of a process to be non – contiguous.

Paging . .

- In paging, the physical memory is divided into fixed-size blocks called page frames, which are the same size as the pages used by the process.
- The process's logical address space is also divided into fixed-size blocks called pages, which are the same size as the page frames.
- When a process requests memory, the operating system allocates one or more page frames to the process and maps the process's logical pages to the physical page frames.

Terminologies Associated with Memory Control

- **Logical Address or Virtual Address:**

  This deal, made by the computer's brain (CPU), helps the computer access its memory. It's like a virtual or digital address because it's not a real spot in memory, but a way to connect to a location in the computer's logical address system.

- **Logical Address Space or Virtual Address Space:**

  This is a collection of logical addresses created by a software program. These addresses are usually shown in words or bytes and are divided into equal-sized sections in a paging system.

- **Physical Address:**

  A "memory address" is like a map that helps the computer find a specific location in its memory. It's the actual code that the memory controller uses to access that memory location.

- **Physical Address Space:**

  This is like a list of physical locations that match up with the computer's logical addresses. These physical locations are typically described in words or bytes and are divided into equal-sized sections in a paging system.

Paging . . .

- The mapping between logical pages and physical page frames is maintained by the page table, which is used by the memory management unit to translate logical addresses into physical addresses.
- The page table maps each logical page number to a physical page frame number.

- In a paging system, we divide the logical address space into fixed-size pages, and each page is connected to a specific place in the physical memory.
- For each running program, there's a page table that links logical addresses to their corresponding physical addresses. When a program needs to access memory, the CPU creates a logical address, which is then transformed into a physical address using the page table.
- The memory controller then uses this physical address to access the correct part of the memory.

Example:

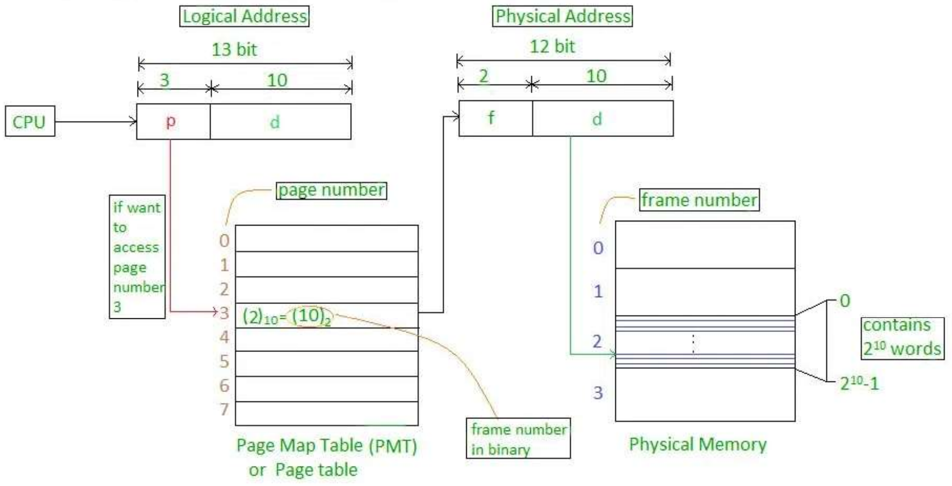
1. If Logical Address = 31 bit, then Logical Address Space = 2^{31} words = 2 G words (1 G = 2^{30})
2. If Logical Address Space = 128 M words = 2^7 * 2^{20} words, then Logical Address = $\log_2 2^{27}$ = 27 bits
3. If Physical Address = 22 bit, then Physical Address Space = 2^{22} words = 4 M words (1 M = 2^{20})
4. If Physical Address Space = 16 M words = 2^4 * 2^{20} words, then Physical Address = $\log_2 2^{24}$ = 24 bits

Let us consider an example:

- Physical Address = 12 bits, then Physical Address Space = 4 K words
- Logical Address = 13 bits, then Logical Address Space = 8 K words
- Page size = frame size = 1 K words (assumption)

Number of frames = Physical Address Space / Frame size = 4 K / 1 K = $\boxed{4}$ = 2^2

Number of pages = Logical Address Space / Page size = 8 K / 1 K = $\boxed{8}$ = 2^3



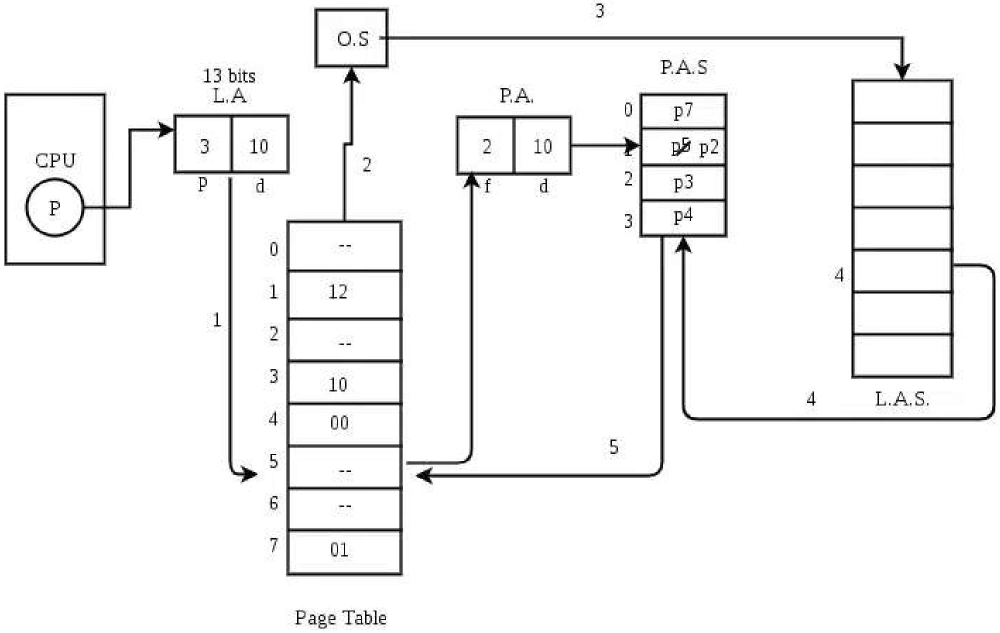
Memory Management Unit (MMU)

The mapping from virtual to physical address is done by the Memory Management Unit (MMU) which is a hardware device and this mapping is known as the paging technique.

- The Physical Address Space is conceptually divided into a number of fixed-size blocks, called **frames**.
- The Logical Address Space is also split into fixed-size blocks, called **pages**.
- Page Size = Frame Size

Demand Paging

- The process of loading the page into memory on demand (whenever a page fault occurs) is known as demand paging. The process includes the following steps are as follows in the next slides:



Demand Paging

Demand Paging. . .

1. If the CPU tries to refer to a page that is currently not available in the main memory, it generates an interrupt indicating a memory access fault.
2. The OS puts the interrupted process in a blocking state. For the execution to proceed the OS must bring the required page into the memory.
3. The OS will search for the required page in the logical address space.
4. The required page will be brought from logical address space to physical address space. The page replacement algorithms are used for the decision-making of replacing the page in physical address space.
5. The page table will be updated accordingly.
6. The signal will be sent to the CPU to continue the program execution and it will place the process back into the ready state.

Segmentation in Operating System

Types of Segmentation in Operating System

1. **Virtual Memory Segmentation:** Each process is divided into a number of segments, but the segmentation is not done all at once. This segmentation may or may not take place at the run time of the program.
2. **Simple Segmentation:** Each process is divided into a number of segments, all of which are loaded into memory at run time, though not necessarily contiguously.

There is no simple relationship between logical addresses and physical addresses in segmentation. A table stores the information about all such segments and is called Segment Table.

Segmentation in Operating System

- A process is divided into Segments.
- The **chunks** that a program is divided into which are not necessarily all of the exact sizes are called segments.
- **Segmentation** gives the user's view of the process which paging does not provide.
- This section discuss how the user's view is mapped to physical memory.

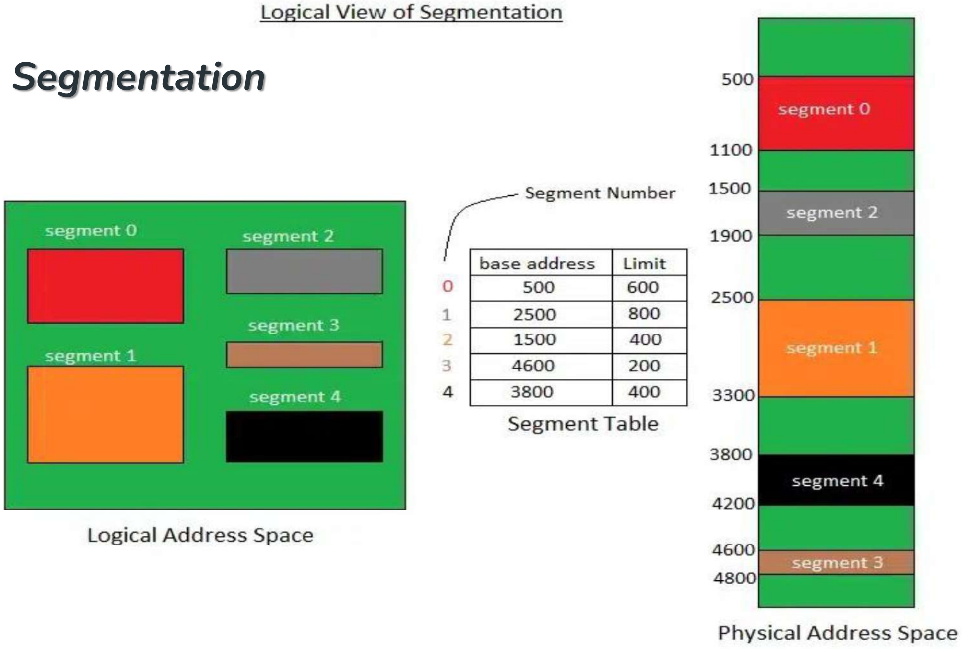
What is Segment Table?

It maps a two-dimensional Logical address into a one-dimensional Physical address. It's each table entry has:

- **Base Address:** It contains the starting physical address where the segments reside in memory.
- **Segment Limit:** Also known as segment offset. It specifies the length of the segment.

Logical View of Segmentation

Segmentation



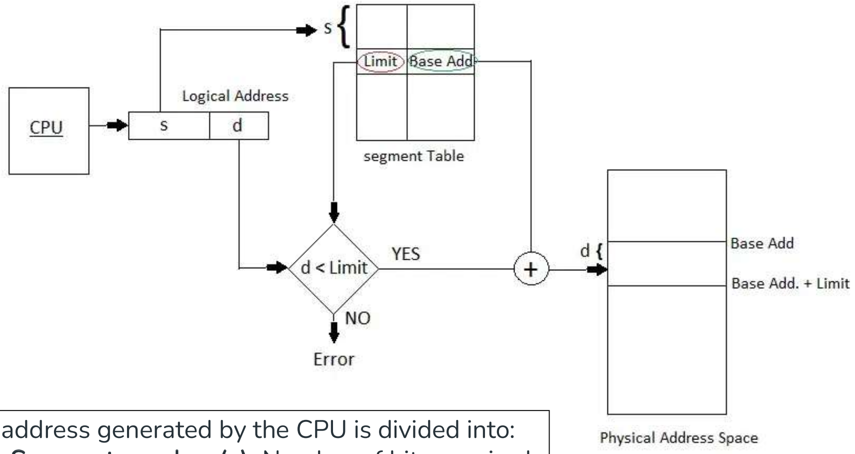
	base address	Limit
0	500	600
1	2500	800
2	1500	400
3	4600	200
4	3800	400

Segment Table

Advantages of Segmentation in Operating System

- No Internal fragmentation.
- Segment Table consumes less space in comparison to Page table in paging.
- As a complete module is loaded all at once, segmentation improves CPU utilization.
- The user’s perception of physical memory is quite similar to segmentation. Users can divide user programs into modules via segmentation. These modules are nothing more than separate processes’ codes.
- The user specifies the segment size, whereas, in paging, the hardware determines the page size.
- Segmentation is a method that can be used to segregate data from security operations.
- **Flexibility:** Segmentation provides a higher degree of flexibility than paging. Segments can be of variable size, and processes can be designed to have multiple segments, allowing for more fine-grained memory allocation.
- **Sharing:** Segmentation allows for sharing of memory segments between processes. This can be useful for inter-process communication or for sharing code libraries.
- **Protection:** Segmentation provides a level of protection between segments, preventing one process from accessing or modifying another process’s memory segment. This can help increase the security and stability of the system.

Translation of Two-dimensional Logical Address to Dimensional Physical Address.

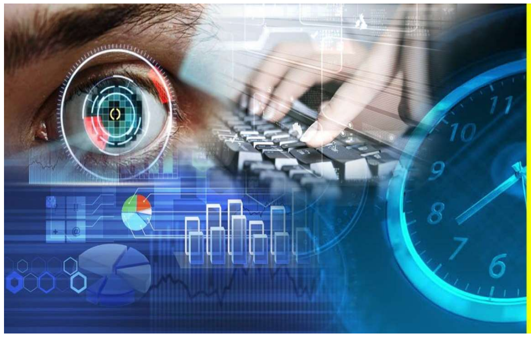


The address generated by the CPU is divided into:

- **Segment number (s):** Number of bits required to represent the segment.
- **Segment offset (d):** Number of bits required to represent the size of the segment.

Disadvantages of Segmentation in Operating System

- As processes are loaded and removed from the memory, the free memory space is broken into little pieces, causing External fragmentation.
- Overhead is associated with keeping a segment table for each activity.
- Due to the need for two memory accesses, one for the segment table and the other for main memory, access time to retrieve the instruction increases.
- **Fragmentation:** As mentioned, segmentation can lead to external fragmentation as memory becomes divided into smaller segments. This can lead to wasted memory and decreased performance.
- **Overhead:** Using a segment table can increase overhead and reduce performance. Each segment table entry requires additional memory, and accessing the table to retrieve memory locations can increase the time needed for memory operations.
- **Complexity:** Segmentation can be more complex to implement and manage than paging. In particular, managing multiple segments per process can be challenging, and the potential for segmentation faults can increase as a result.



HNDIT 3052 Operating Systems

File Systems

What is file system in operating system?

- An integral part of an OS is what is called a File System. A File System is a data structure that stores data and information on storage devices (hard drives, floppy disc, etc.), making them easily retrievable. Different OS's use different file systems, but all have similar features.

File Systems

Topic:

- File Systems

Subtopics:

- Files:
  1. File Naming,
  2. structure ,
  3. Types,
  4. Access ,
  5. Attributes,
  6. Operations
- **Directories**
- **file System Implementation**
- **File system Management and optimization**

File Systems in Operating System

A file system is a method an operating system uses to store, organize, and manage files and directories on a storage device. Some common types of file systems include:

1. **FAT (File Allocation Table):** An older file system used by older versions of Windows and other operating systems.
2. **NTFS (New Technology File System):** A modern file system used by Windows. It supports features such as file and folder permissions, compression, and encryption.
3. **ext (Extended File System):** A file system commonly used on Linux and Unix-based operating systems.
4. **HFS (Hierarchical File System):** A file system used by macOS.
5. **APFS (Apple File System):** A new file system introduced by Apple for their Macs and iOS devices.

The advantages of using a file system

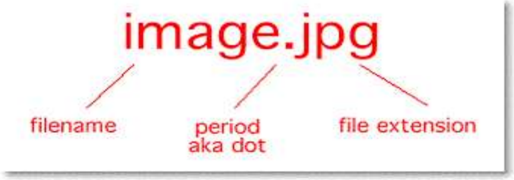
- **Organization:** A file system allows files to be organized into directories and subdirectories, making it easier to manage and locate files.
- **Data protection:** File systems often include features such as file and folder permissions, backup and restore, and error detection and correction, to protect data from loss or corruption.
- **Improved performance:** A well-designed file system can improve the performance of reading and writing data by organizing it efficiently on disk.

A file is a collection of related information that is recorded on secondary storage. Or file is a collection of logically related entities. From the user's perspective, a file is the smallest allotment of logical secondary storage.

Disadvantages of using a file system

- **Compatibility issues:** Different file systems may not be compatible with each other, making it difficult to transfer data between different operating systems.
- **Disk space overhead:** File systems may use some disk space to store metadata and other overhead information, reducing the amount of space available for user data.
- **Vulnerability:** File systems can be vulnerable to data corruption, malware, and other security threats, which can compromise the stability and security of the system.

- **(1) File Naming:**
- **The name of the file is divided into two parts as shown below:**
  1. name
  2. extension, separated by a period.

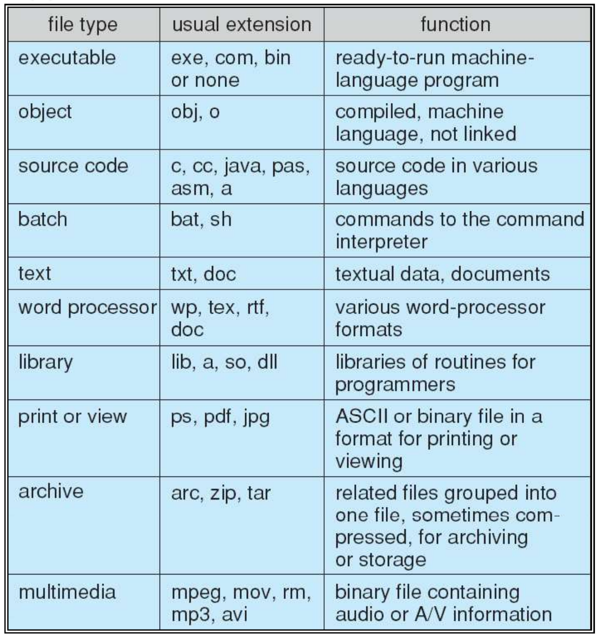


What are file extensions?

- **File extensions** are the end part of file names, the part that comes after a dot and is usually 3 or 4 characters in length.
- Examples of web file extensions:
  - Web pages: **.html .htm .php .jsp**
    - Example: index.html
  - Web image file types: **.png .jpg .svg .gif**
    - Example: logo.png
  - CSS and JavaScript files: **.css .js**
    - Example: global-styles.css
  - Other file extension:
    - MS Word: **.docx**
      - Example: report.docx
    - Excel: **.xlsx**
      - Example: budget2019.xlsx

- By default, file extensions are usually not visible in Windows and, sometimes, Macs.
- You can change settings to make extensions visible.
- The benefit of seeing file extensions:
  - If you are able to see the extension of a file you will probably be able to determine what the file is used for (especially if the file name is meaningful) and which programs(s) can open it.
- We do not need to bother:
  - It is beneficial to be able to see file extensions because when you see one it enables you to rapidly identify the file-type without needing to open the file.
  - Combining the name of a file with its file extension may be all that you need to know precisely what the purpose of the file is.
- NOTE: changing a file extension will not change the file into another file type nor have the characteristics of that other file type.
  - For example, changing an Excel document extension from .xlsx to .jpg will not allow it to be displayed in a web browser. The .xlsx file type cannot be displayed by a web browser regardless of its file extension.

File Types, names and extensions



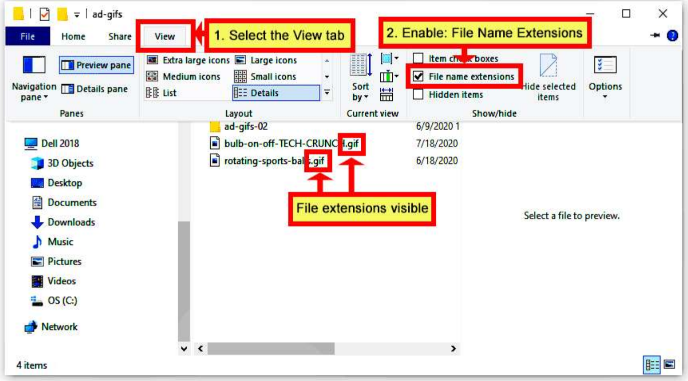
file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual 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 compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

Setting Windows to display file extensions

Windows 10 - my instructions

- Open the Windows 10 **File Explorer**
- Click on the **View** tab.
- Enable **File name extensions**
- File extensions for individual files ought to be visible now. Note that folders do not have file extensions.

Windows 10 File Explorer



1. Select the View tab

2. Enable: File Name Extensions

File extensions visible

☑ File name extensions

ad-gifs-02 6/9/2020 1

bulb-on-off-TECH-CRUNCH.gif 7/18/2020

rotating-sports-balls.gif 6/18/2020

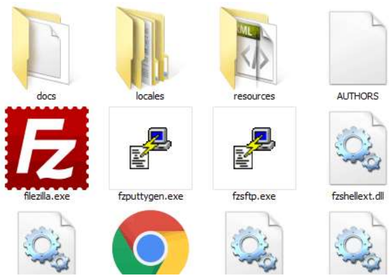
Select a file to preview.

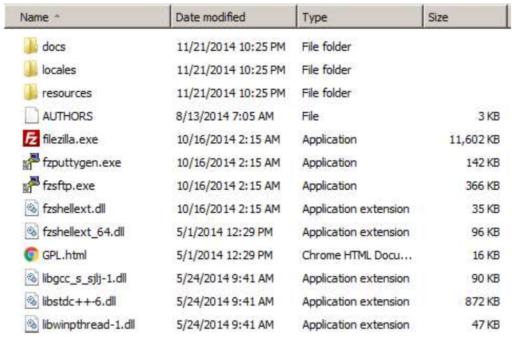
4 items

Setting Windows to display files and folders in "details" view

You can set Windows to provide many types of information about files and folders ("details"):

1. Open **Windows Explorer** (or, **File Explorer** on Windows 10).
2. Right click on the folder whose you are interested in using or learning about.
3. Select **Properties**
4. Click on **Customize** tab.
5. Beneath **Optimize this folder for:** select **Documents**
6. Enable (check) the checkbox **Also apply this template to all sub-folders**
7. Click **Apply**
8. Click **OK**





Name	Date modified	Type	Size
docs	11/21/2014 10:25 PM	File folder	
locales	11/21/2014 10:25 PM	File folder	
resources	11/21/2014 10:25 PM	File folder	
AUTHORS	8/13/2014 7:05 AM	File	3 KB
filezilla.exe	10/16/2014 2:15 AM	Application	11,602 KB
fzputtygen.exe	10/16/2014 2:15 AM	Application	142 KB
fzsftp.exe	10/16/2014 2:15 AM	Application	366 KB
fzshellext.dll	10/16/2014 2:15 AM	Application extension	35 KB
fzshellext_64.dll	5/1/2014 12:29 PM	Application extension	96 KB
GPL.html	5/1/2014 12:29 PM	Chrome HTML Docu...	16 KB
libgcc_s_sjlj-1.dll	5/24/2014 9:41 AM	Application extension	90 KB
libstdc++-6.dll	5/24/2014 9:41 AM	Application extension	872 KB
libwinpthread-1.dll	5/24/2014 9:41 AM	Application extension	47 KB

Comparison of Large Icons and Details views

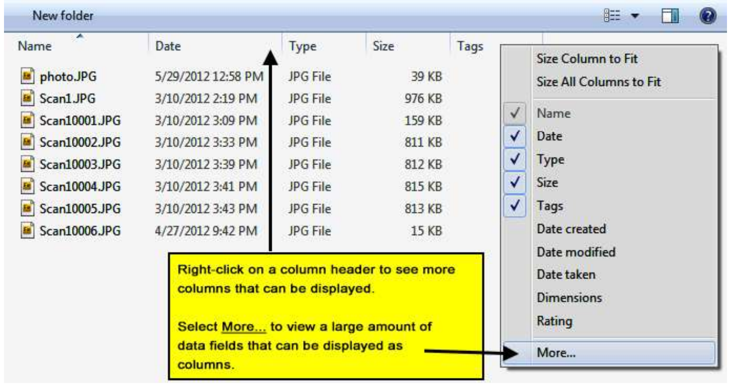
1. Icons view

- Provides almost no information other than file and folder names.
- The icons themselves can be deceptive. In the image below, the Google Chrome icon (partially shown) is used to represent an HTML file. The icon is being shown because I manually set HTML files to be opened in Chrome when the file is double-clicked. However, the Chrome icon suggests that it is only for use in Chrome or that it is some type of Google file.

2. Details view

- Details view provides a view similar to a spreadsheet or database list.
- You can sort by any of the data fields (columns) by clicking on the column header. Click a second time to reverse the sort.
- By default, you are provided with the data fields (columns) containing the name, type of object, and, for files, the file size.

- You can add other columns/fields by right-clicking on any of the column headers and selecting from the menu of options (see image below).But note that the database fields that can be associated with a file varies depending on the file type that it is.
- For example, music files and Microsoft Word files have many fields that are editable.
- One field that *is* available for images is **dimensions**, and I recommend that you display that column if you work with images.

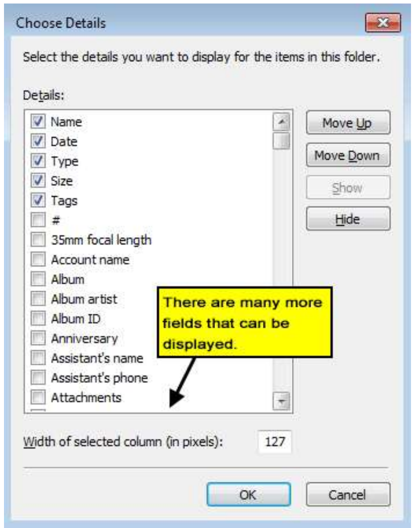


Name	Date	Type	Size	Tags
photo.JPG	5/29/2012 12:58 PM	JPG File	39 KB	
Scan1.JPG	3/10/2012 2:19 PM	JPG File	976 KB	
Scan10001.JPG	3/10/2012 3:09 PM	JPG File	159 KB	
Scan10002.JPG	3/10/2012 3:33 PM	JPG File	811 KB	
Scan10003.JPG	3/10/2012 3:39 PM	JPG File	812 KB	
Scan10004.JPG	3/10/2012 3:41 PM	JPG File	815 KB	
Scan10005.JPG	3/10/2012 3:43 PM	JPG File	813 KB	
Scan10006.JPG	4/27/2012 9:42 PM	JPG File	15 KB	

Right-click on a column header to see more columns that can be displayed.

Select More... to view a large amount of data fields that can be displayed as columns.

- Click on the **More...** menu option (*above, right*) and you will be shown an astounding array of fields that you can display in the List view. But, again, note that the data fields that can be saved with a file varies with file type.



Choose Details

Select the details you want to display for the items in this folder.

Details:

- ☑ Name
- ☑ Date
- ☑ Type
- ☑ Size
- ☑ Tags
- ☐ #
- ☐ 35mm focal length
- ☐ Account name
- ☐ Album
- ☐ Album artist
- ☐ Album ID
- ☐ Anniversary
- ☐ Assistant's name
- ☐ Assistant's phone
- ☐ Attachments

Move Up | Move Down | Show | Hide

Width of selected column (in pixels): 127

OK | Cancel

There are many more fields that can be displayed.

•Note that if you right-click on a file and select **Properties** you will see the data and field names associated with that type of file. If the data values are editable, you can click on the value and enter or edit the value.

(2) File System Structure

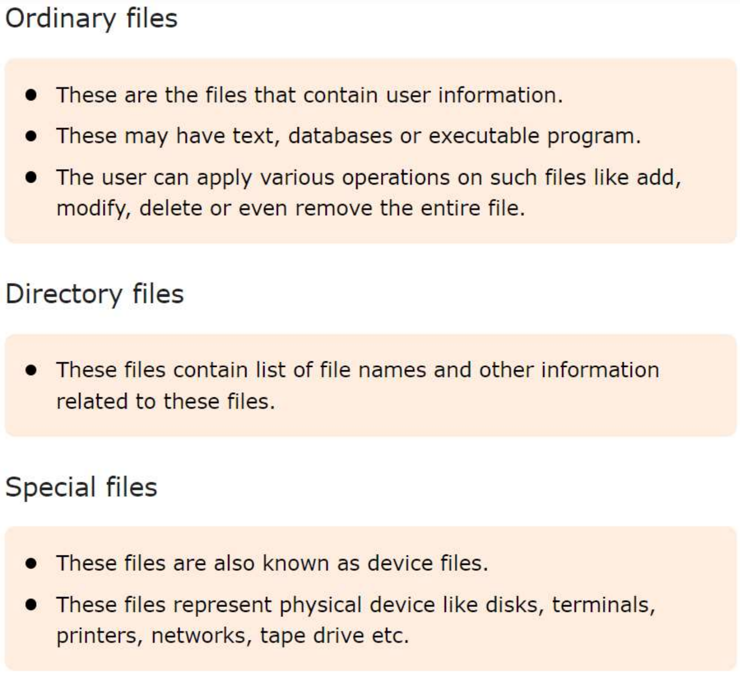
- **File:** A file is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks. In general, a file is a sequence of bits, bytes, lines or records whose meaning is defined by the files creator and user.
- **File Structure**: A File Structure should be according to a required format that the operating system can understand.

- A file has a certain defined structure according to its type.
- A text file is a sequence of characters organized into lines.
- A source file is a sequence of procedures and functions.
- An object file is a sequence of bytes organized into blocks that are understandable by the machine.
- When operating system defines different file structures, it also contains the code to support these file structure. Unix, MS-DOS support minimum number of file structure.

Following Issues Are Handled By The File System

- A variety of data structures where the file could be kept. The file system's job is to keep the files organized in the best way possible.
- A free space is created on the hard drive whenever a file is deleted from it. To reallocate them to other files, many of these spaces may need to be recovered.
- Choosing where to store the files on the hard disc is the main issue with files one block may or may not be used to store a file. It may be kept in the disk's **non-contiguous** blocks.
- We must keep track of all the blocks where the files are partially located.

- **File Type:** File type refers to the ability of the operating system to distinguish different types of file such as text files source files and binary files etc. Many operating systems support many types of files. Operating system like MS-DOS and UNIX have the following types of files –



Ordinary files

- These are the files that contain user information.
- These may have text, databases or executable program.
- The user can apply various operations on such files like add, modify, delete or even remove the entire file.

Directory files

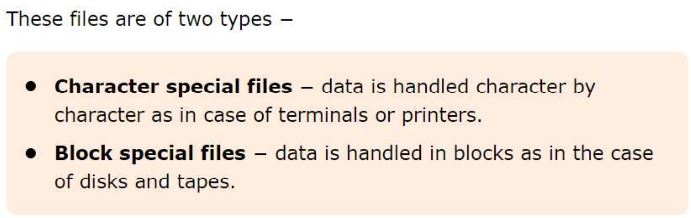
- These files contain list of file names and other information related to these files.

Special files

- These files are also known as device files.
- These files represent physical device like disks, terminals, printers, networks, tape drive etc.

(5). Files Attributes: A file has a name and data. Moreover, it also stores meta information like file creation date and time, current size, last modified date, etc. All this information is called the attributes of a file system. Here, are some **important File attributes** used in OS:

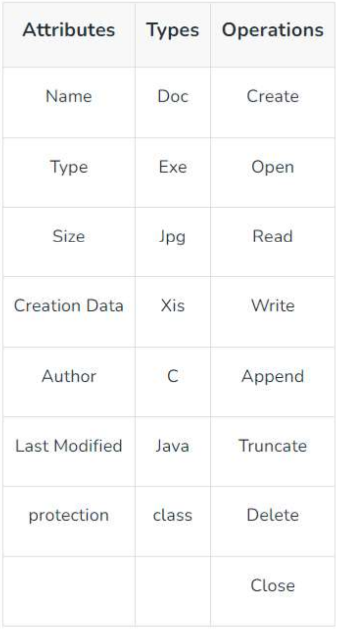
- **Name:** It is the only information stored in a human-readable form. Every file carries a name by which the file is recognized in the file system. One directory cannot have two files with the same name.
- **Identifier**: Every file is identified by a unique tag number within a file system known as an identifier. Along with the name, Each File has its own extension which identifies the type of the file. For example, a text file has the extension **.txt,** A video file can have the extension **.mp4.**
- **Location:** Points to file location on device. In the File System, there are several locations on which, the files can be stored. Each file carries its location as its attribute.
- **Type:** This attribute is required for systems that support various types of files. In a File System, the Files are classified in different types such as video files, audio files, text files, executable files, etc.
- **Size**. Attribute used to display the current file size. The Size of the File is one of its most important attribute. By size of the file, we mean the number of bytes acquired by the file in the memory.
- **Protection**. This attribute assigns and controls the access rights of reading, writing, and executing the file. The Admin of the computer may want the different protections for the different files. Therefore each file carries its own set of permissions to the different group of Users.
- **Time, date and security:** It is used for protection, security, and also used for monitoring. (Time & Date: Every file carries a time stamp which contains the time and date on which the file is last modified.)



These files are of two types –

- **Character special files** – data is handled character by character as in case of terminals or printers.
- **Block special files** – data is handled in blocks as in the case of disks and tapes.

Files Attributes And Their Operations:



Attributes	Types	Operations
Name	Doc	Create
Type	Exe	Open
Size	Jpg	Read
Creation Data	Xis	Write
Author	C	Append
Last Modified	Java	Truncate
protection	class	Delete
		Close

(6). Operations on the File

- A file is a collection of logically related data that is recorded on the secondary storage in the form of sequence of operations. The content of the files are defined by its creator who is creating the file. The various operations which can be implemented on a file such as read, write, open and close etc. are called file operations. These operations are performed by the user by using the commands provided by the operating system. Some common operations are as follows:

  1. **Create operation:**
  2. **Open operation:**
  3. **Write operation:**
  4. **Read operation:**
  5. **Re-position or Seek operation:**
  6. **Delete operation:**
  7. **Truncate operation:**
  8. **Close operation:**
  9. **Append operation:**
  10. **Rename operation:**

File Operations:

1.Create operation: This operation is used to create a file in the file system. It is the most widely used operation performed on the file system. To create a new file of a particular type the associated application program calls the file system. This file system allocates space to the file. As the file system knows the format of directory structure, so entry of this new file is made into the appropriate directory.

2. Open operation: This operation is the common operation performed on the file. Once the file is created, it must be opened before performing the file processing operations. When the user wants to open a file, it provides a file name to open the particular file in the file system. It tells the operating system to invoke the open system call and passes the file name to the file system.

3. Write operation: This operation is used to write the information into a file. A system call write is issued that specifies the name of the file and the length of the data has to be written to the file. Whenever the file length is increased by specified value and the file pointer is repositioned after the last byte written.

4. Read operation: This operation reads the contents from a file. A Read pointer is maintained by the OS, pointing to the position up to which the data has been read.

5. Re-position or Seek operation: The seek system call re-positions the file pointers from the current position to a specific place in the file i.e. forward or backward depending upon the user's requirement. This operation is generally performed with those file management systems that support direct access files.

File Operations:

6. Delete operation: Deleting the file will not only delete all the data stored inside the file it is also used so that disk space occupied by it is freed. In order to delete the specified file the directory is searched. When the directory entry is located, all the associated file space and the directory entry is released.

7. Truncate operation: Truncating is simply deleting the file except deleting attributes. The file is not completely deleted although the information stored inside the file gets replaced.

8. Close operation: When the processing of the file is complete, it should be closed so that all the changes made permanent and all the resources occupied should be released. On closing it deallocates all the internal descriptors that were created when the file was opened.

9. Append operation: This operation adds data to the end of the file.

10. Rename operation: This operation is used to rename the existing file.

Functions of File:

- Create file, find space on disk, and make an entry in the directory.
- Write to file, requires positioning within the file
- Read from file involves positioning within the file
- Delete directory entry, regain disk space.
- Reposition: move read/write position.

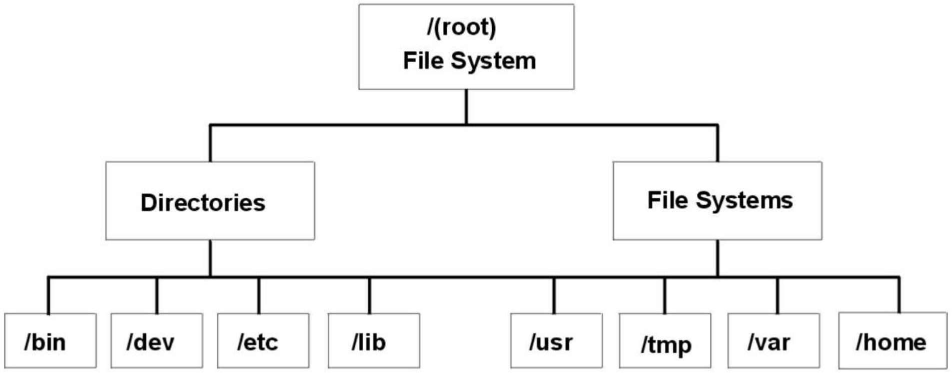
Directories

- It is important to understand the ***difference between*** a **file system** and a **directory**. A file system is a section of hard disk that has been allocated to contain files. This section of hard disk is accessed by mounting the file system over a directory. After the file system is mounted, it looks just like any other directory to the end user.
- Because of the structural differences between the file systems and directories, the data within these entities can be managed separately.
- When the operating system is installed for the first time, it is loaded into a directory structure, as shown in the following illustration.

The file tree has the following characteristics:

- Files that can be shared by machines of the same hardware architecture are located in the /usr file system.
- Variable per-client files, for example, spool and mail files, are located in the /var file system.
- The /(root) file system contains files and directories critical for system operation. For example, it contains
  - A device directory (/dev)
  - Mount points where file systems can be mounted onto the root file system, for example, /mnt
- The /home file system is the mount point for users' home directories.
- For servers, the /export directory contains paging-space files, per-client (unshared) root file systems, dump, home, and /usr/share directories for diskless clients, as well as exported /usr directories.
- The /proc file system contains information about the state of processes and threads in the system.
- The /opt file system contains optional software, such as applications.

- *Consider a following Linux file system.*
- *Figure : / (root) File System Tree. This tree chart shows a directory structure with the / (root) file system at the top, branching downward to directories and file systems. Directories branch to /bin, /dev, /etc, and /lib. File systems branch to /usr, /tmp, /var, and /home.*

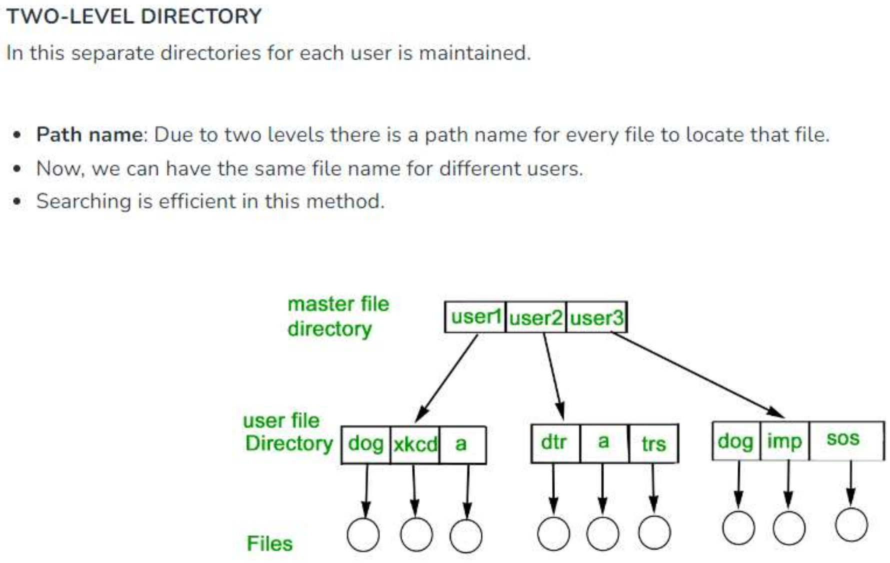


The directories on the right (/usr, /tmp, /var, and /home) are all file systems so they have separate sections of the hard disk allocated for their use. These file systems are mounted automatically when the system is started, so the end user does not see the difference between these file systems and the directories listed

FILE DIRECTORIES

- The collection of files is a file directory. The directory contains information about the files, including attributes, location, and ownership. Much of this information, especially that is concerned with storage, is managed by the operating system. The directory is itself a file, accessible by various file management routines.
- **Information contained in a device directory is:**
  1. Name
  2. Type
  3. Address
  4. Current length
  5. Maximum length
  6. Date last accessed
  7. Date last updated
  8. Owner id
  9. Protection information

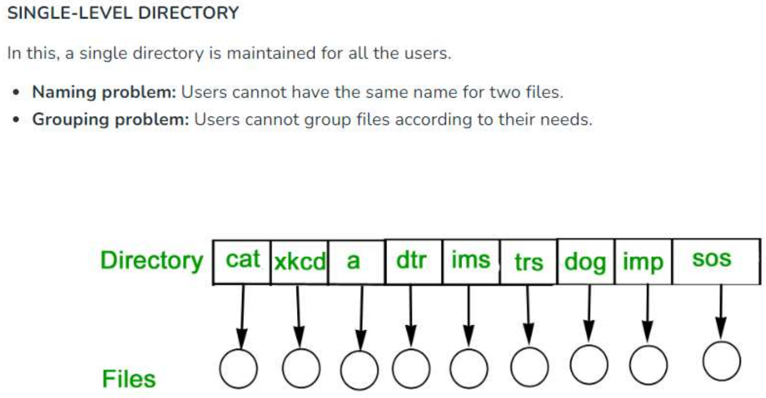
- **The operation performed on the directory are:**
  1. Search for a file
  2. Create a file
  3. Delete a file
  4. List a directory
  5. Rename a file
  6. Traverse the file system
- **The advantages of maintaining directories are:**
  1. **Efficiency:** A file can be located more quickly.
  2. **Naming:** It becomes convenient for users as two users can have same name for different files or may have different name for same file.
  3. **Grouping:** Logical grouping of files can be done by properties e.g. all java programs, all games etc.



TWO-LEVEL DIRECTORY

In this separate directories for each user is maintained.

- **Path name**: Due to two levels there is a path name for every file to locate that file.
- Now, we can have the same file name for different users.
- Searching is efficient in this method.



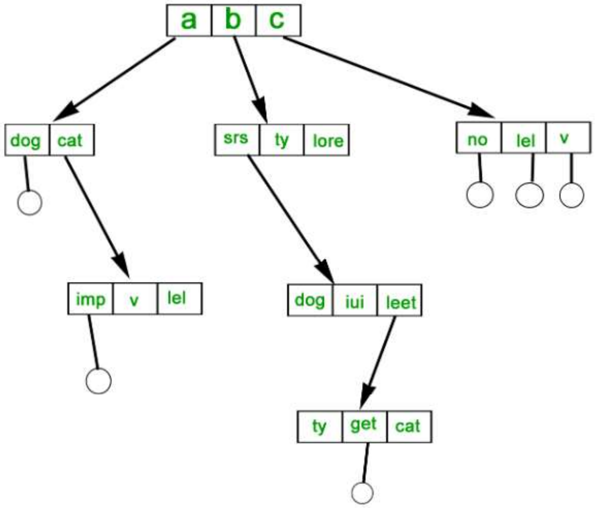
SINGLE-LEVEL DIRECTORY

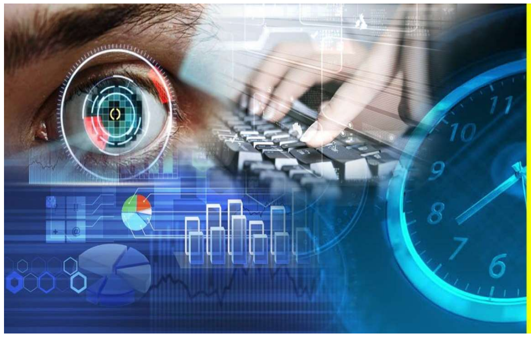
In this, a single directory is maintained for all the users.

- **Naming problem:** Users cannot have the same name for two files.
- **Grouping problem:** Users cannot group files according to their needs.

TREE-STRUCTURED DIRECTORY:

The directory is maintained in the form of a tree. Searching is efficient and also there is grouping capability. We have absolute or relative path name for a file.





HNDIT 3052 Operating Systems

Input / Output management and Disk scheduling

Input / Output Management

- Input/output (I/O) management involves controlling the flow of data into and out of a data processing complex.
- Controlling all of a computer's input and output devices is one of the primary or main functions of an OS, or operating system.
- The operating system must issue the commands to the input and output devices, catch the interrupts, and handle the errors.
- In addition, the operating system should act as a bridge between the input/output devices and the rest of the computer system. This interface should be straightforward, device-independent (where possible), and simple to use.
- In any computer system, there are far too many input/output devices, such as discs, keyboards, monitors, and so on.

Input / Output management and Disk scheduling

Subtopics:

1. Principles of I/O Hardware
2. Principles of I/O Software
3. Disc Hardware
4. Disk Formatting
5. Disk Arm Scheduling Algorithms

1. Principles of I/O Hardware

- One of the important jobs of an Operating System is to manage various I/O devices including mouse, keyboards, touch pad, disk drives, display adapters, USB devices, Bit-mapped screen, LED, Analog-to-digital converter, On/off switch, network connections, audio I/O, printers etc.
- An I/O system is required to take an application I/O request and send it to the physical device, then take whatever response comes back from the device and send it to the application. I/O devices can be divided into two categories –

i. **Block devices:** A block device is one with which the driver communicates by sending entire blocks of data. For example, Hard disks, USB cameras, Disk-On-Key etc.

ii. **Character devices:** A character device is one with which the driver communicates by sending and receiving single characters (bytes, octets). For example, serial ports, parallel ports, sounds cards etc

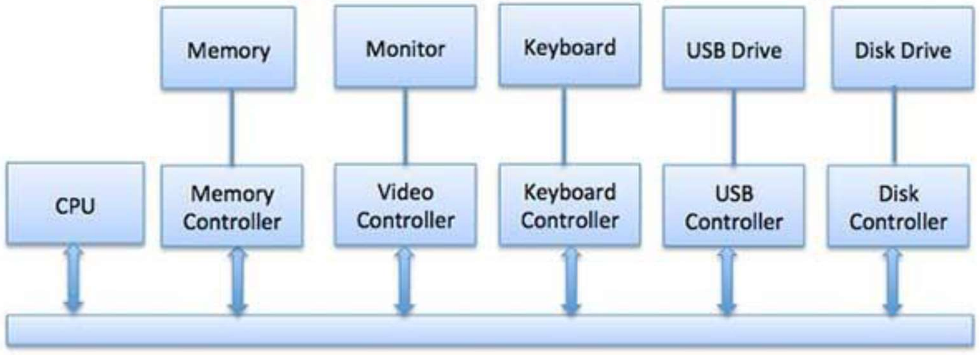
Device Controllers

- Device drivers are software modules that can be plugged into an OS to handle a particular device. Operating System takes help from device drivers to handle all I/O devices.
- The Device Controller works like an interface between a device and a device driver. I/O units (Keyboard, mouse, printer, etc.) typically consist of a mechanical component and an electronic component where electronic component is called the device controller.
- There is always a device controller and a device driver for each device to communicate with the Operating Systems. A device controller may be able to handle multiple devices. As an interface its main task is to convert serial bit stream to block of bytes, perform error correction as necessary.

Synchronous vs asynchronous I/O

- **Synchronous I/O** – In this scheme CPU execution waits while I/O proceeds
- **Asynchronous I/O** – I/O proceeds concurrently with CPU execution

Any device connected to the computer is connected by a plug and socket, and the socket is connected to a device controller. Following is a model for connecting the CPU, memory, controllers, and I/O devices where CPU and device controllers all use a common bus for communication.



Communication to I/O Devices:

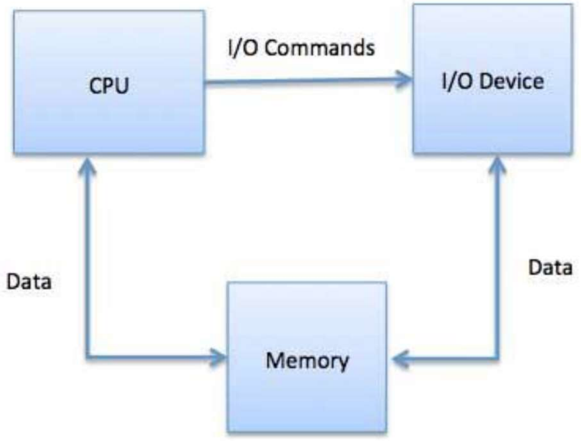
- The CPU must have a way to pass information to and from an I/O device. There are three approaches available to communicate with the CPU and Device.
  - i. Special Instruction I/O
  - ii. Memory-mapped I/O
  - iii. Direct memory access (DMA)

i. Special Instruction I/O

- This uses CPU instructions that are specifically made for controlling I/O devices. These instructions typically allow data to be sent to an I/O device or read from an I/O device.

ii. Memory-mapped I/O

- When using memory-mapped I/O, the same address space is shared by memory and I/O devices. The device is connected directly to certain main memory locations so that I/O device can transfer block of data to/from memory without going through CPU.

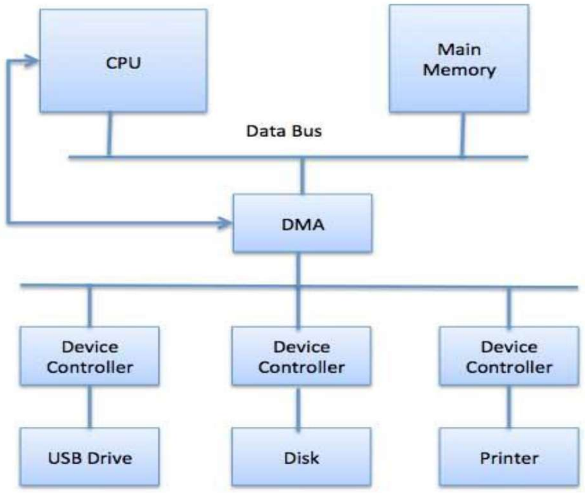


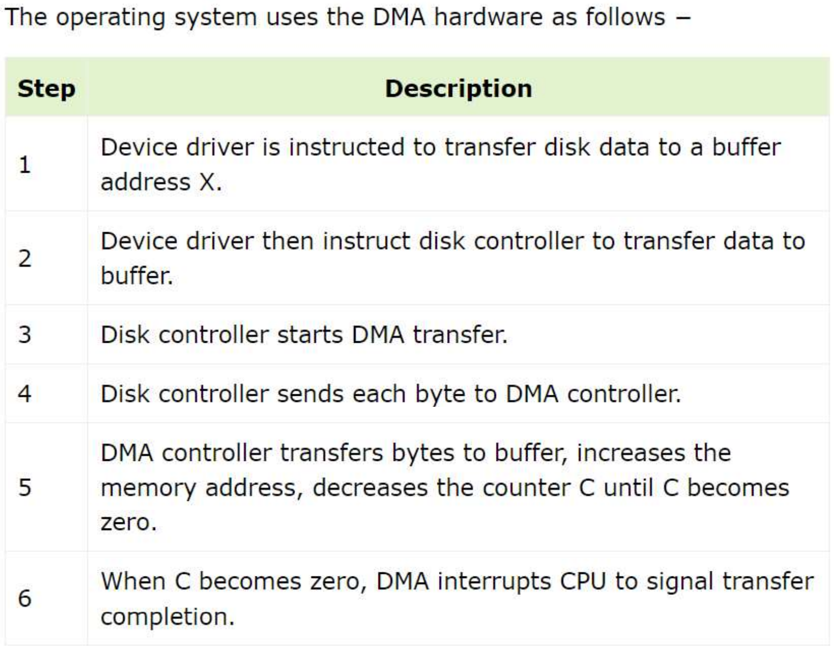
iii. Direct Memory Access (DMA)

- Slow devices like keyboards will generate an interrupt to the main CPU after each byte is transferred. If a fast device such as a disk generated an interrupt for each byte, the operating system would spend most of its time handling these interrupts. So a typical computer uses direct memory access (DMA) hardware to reduce this overhead.
- Direct Memory Access (DMA) means CPU grants I/O module authority to read from or write to memory without involvement. DMA module itself controls exchange of data between main memory and the I/O device. CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred.

- While using memory mapped IO, OS allocates buffer in memory and informs I/O device to use that buffer to send data to the CPU. I/O device operates asynchronously with CPU, interrupts CPU when finished.
- The advantage to this method is that every instruction which can access memory can be used to manipulate an I/O device. Memory mapped IO is used for most high-speed I/O devices like disks, communication interfaces.

- Direct Memory Access needs a special hardware called DMA controller (DMAC) that manages the data transfers and arbitrates access to the system bus. The controllers are programmed with source and destination pointers (where to read/write the data), counters to track the number of transferred bytes, and settings, which includes I/O and memory types, interrupts and states for the CPU cycles.



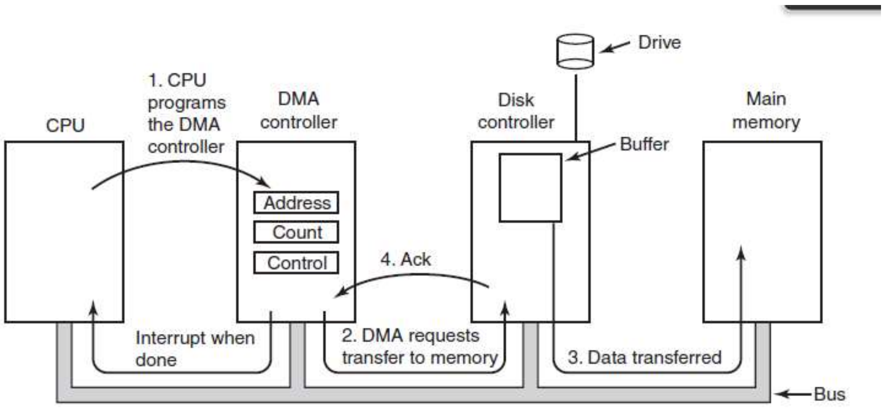


The operating system uses the DMA hardware as follows –

Step	Description
1	Device driver is instructed to transfer disk data to a buffer address X.
2	Device driver then instruct disk controller to transfer data to buffer.
3	Disk controller starts DMA transfer.
4	Disk controller sends each byte to DMA controller.
5	DMA controller transfers bytes to buffer, increases the memory address, decreases the counter C until C becomes zero.
6	When C becomes zero, DMA interrupts CPU to signal transfer completion.

Polling vs Interrupts I/O:

- A computer must have a way of detecting the arrival of any type of input. There are two ways that this can happen, known as **polling** and **interrupts**. Both of these techniques allow the processor to deal with events that can happen at any time and that are not related to the process it is currently running.



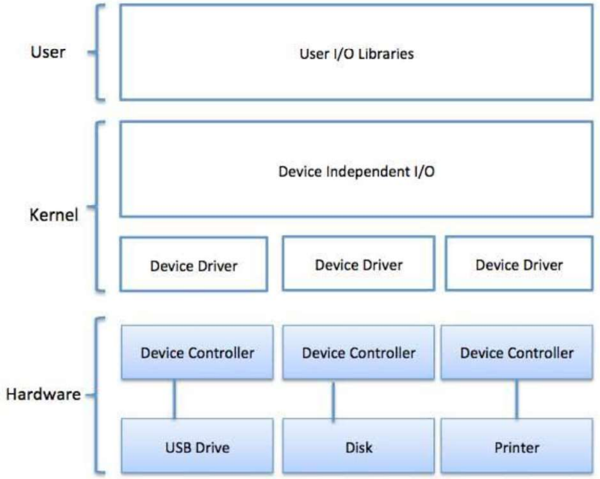
Polling I/O:

- Polling is the simplest way for an I/O device to communicate with the processor. The process of periodically checking status of the device to see if it is time for the next I/O operation, is called polling. The I/O device simply puts the information in a Status register, and the processor must come and get the information.
- Most of the time, devices will not require attention and when one does it will have to wait until it is next interrogated by the polling program. This is an inefficient method and much of the processors time is wasted on unnecessary polls.
- Compare this method to a teacher continually asking every student in a class, one after another, if they need help. Obviously the more efficient method would be for a student to inform the teacher whenever they require assistance.

Interrupts I/O:

- An alternative scheme for dealing with I/O is the interrupt-driven method. An interrupt is a signal to the microprocessor from a device that requires attention.
- A device controller puts an interrupt signal on the bus when it needs CPU's attention when CPU receives an interrupt, It saves its current state and invokes the appropriate interrupt handler using the interrupt vector (addresses of OS routines to handle various events). When the interrupting device has been dealt with, the CPU continues with its original task as if it had never been interrupted.

- A key concept in the design of I/O software is that it should be device independent where it should be possible to write programs that can access any I/O device without having to specify the device in advance. For example, a program that reads a file as input should be able to read a file on a floppy disk, on a hard disk, or on a CD-ROM, without having to modify the program for each different device.



2. Principles of I/O Software

I/O software is often organized in the following layers –

- **User Level Libraries** – This provides simple interface to the user program to perform input and output. For example, **stdio** is a library provided by C and C++ programming languages.
- **Kernel Level Modules** – This provides device driver to interact with the device controller and device independent I/O modules used by the device drivers.
- **Hardware** – This layer includes actual hardware and hardware controller which interact with the device drivers and makes hardware alive.

Device Drivers:

- Device drivers are software modules that can be plugged into an OS to handle a particular device. Operating System takes help from device drivers to handle all I/O devices. Device drivers encapsulate device-dependent code and implement a standard interface in such a way that code contains device-specific register reads/writes. Device driver, is generally written by the device's manufacturer and delivered along with the device on a CD-ROM.

A device driver performs the following jobs –

  - To accept request from the device independent software above to it.
  - Interact with the device controller to take and give I/O and perform required error handling
  - Making sure that the request is executed successfully

- How a device driver handles a request is as follows: Suppose a request comes to read a block N. If the driver is idle at the time a request arrives, it starts carrying out the request immediately. Otherwise, if the driver is already busy with some other request, it places the new request in the queue of pending requests.

Interrupt handlers:

- An interrupt handler, also known as an interrupt service routine or ISR, is a piece of software or more specifically a callback function in an operating system or more specifically in a device driver, whose execution is triggered by the reception of an interrupt.
- When the interrupt happens, the interrupt procedure does whatever it has to in order to handle the interrupt, updates data structures and wakes up process that was waiting for an interrupt to happen.
- The interrupt mechanism accepts an address – a number that selects a specific interrupt handling routine/function from a small set. In most architectures, this address is an offset stored in a table called the interrupt vector table. This vector contains the memory addresses of specialized interrupt handlers.

User-Space I/O Software:

- These are the libraries which provide richer and simplified interface to access the functionality of the kernel or ultimately interactive with the device drivers. Most of the user-level I/O software consists of library procedures with some exception like spooling system which is a way of dealing with dedicated I/O devices in a multiprogramming system.
- I/O Libraries (e.g., stdio) are in user-space to provide an interface to the OS resident device-independent I/O SW. For example putchar(), getchar(), printf() and scanf() are example of user level I/O library stdio available in C programming.

Device-Independent I/O Software

- The basic function of the device-independent software is to perform the I/O functions that are common to all devices and to provide a uniform interface to the user-level software. Though it is difficult to write completely device independent software but we can write some modules which are common among all the devices. Following is a list of functions of device-independent I/O Software –
  - Uniform interfacing for device drivers
  - Device naming - Mnemonic names mapped to Major and Minor device numbers
  - Device protection
  - Providing a device-independent block size
  - Buffering because data coming off a device cannot be stored in final destination.
  - Storage allocation on block devices
  - Allocation and releasing dedicated devices
  - Error Reporting

Kernel I/O Subsystem:

Kernel I/O Subsystem is responsible to provide many services related to I/O. Following are some of the services provided.

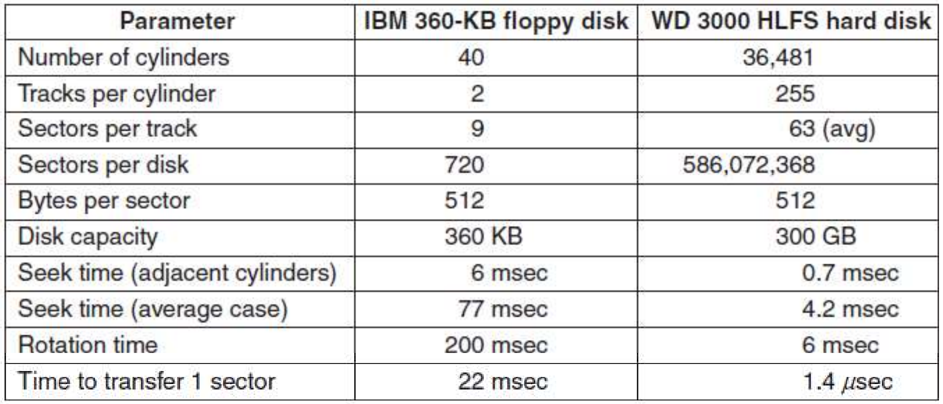
- **Scheduling** – Kernel schedules a set of I/O requests to determine a good order in which to execute them. When an application issues a blocking I/O system call, the request is placed on the queue for that device. The Kernel I/O scheduler rearranges the order of the queue to improve the overall system efficiency and the average response time experienced by the applications.
- **Buffering** – Kernel I/O Subsystem maintains a memory area known as **buffer** that stores data while they are transferred between two devices or between a device with an application operation. Buffering is done to cope with a speed mismatch between the producer and consumer of a data stream or to adapt between devices that have different data transfer sizes.
- **Caching** – Kernel maintains cache memory which is region of fast memory that holds copies of data. Access to the cached copy is more efficient than access to the original.
- **Spooling and Device Reservation** – A spool is a buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams. The spooling system copies the queued spool files to the printer one at a time. In some operating systems, spooling is managed by a system daemon process. In other operating systems, it is handled by an in kernel thread.
- **Error Handling** – An operating system that uses protected memory can guard against many kinds of hardware and application errors.

3. Disc Hardware

- Disks come in a variety of types. The most common ones are the **magnetic hard disks**. They are characterized by the fact that reads and writes are equally fast, which makes them suitable as secondary memory (paging, file systems, etc.).
- Arrays of these disks are sometimes used to provide highly reliable storage. For distribution of programs, data, and movies, optical disks (DVDs and Blu-ray) are also important. Finally, solid-state disks are increasingly popular as they are fast and do not contain moving parts. In the following sections we will discuss magnetic disks as an example of the hardware and then describe the software for disk devices in general.

Disk parameters:

- Disk parameters for the original IBM PC 360-KB floppy disk and a Western Digital WD 3000 HLFS (‘‘Velociraptor’’) hard disk.



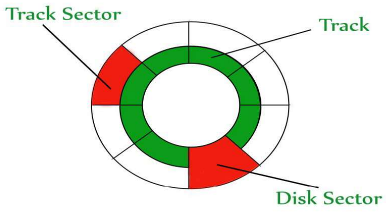
Parameter	IBM 360-KB floppy disk	WD 3000 HLFS hard disk
Number of cylinders	40	36,481
Tracks per cylinder	2	255
Sectors per track	9	63 (avg)
Sectors per disk	720	586,072,368
Bytes per sector	512	512
Disk capacity	360 KB	300 GB
Seek time (adjacent cylinders)	6 msec	0.7 msec
Seek time (average case)	77 msec	4.2 msec
Rotation time	200 msec	6 msec
Time to transfer 1 sector	22 msec	1.4 μsec

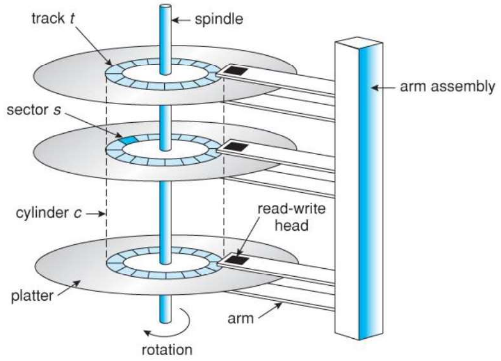
Magnetic Hard Disks:

- Magnetic disks are organized into cylinders, each one containing as many tracks as there are heads stacked vertically. The tracks are divided into sectors, with the number of sectors around the circumference typically being 8 to 32 on floppy disks, and up to several hundred on hard disks. The number of heads varies from 1 to about 16.
- Older disks have little electronics and just deliver a simple serial bit stream. On these disks, the controller does most of the work. On other disks, in particular, **IDE** (**Integrated Drive Electronics**) and **SATA** (**Serial ATA**) disks, the disk drive itself contains a microcontroller that does considerable work and allows the real controller to issue a set of higher-level commands. The controller often does track caching, bad-block remapping, and much more.

What is a Hard Disk Drive?

- A hard disk is a memory storage device that looks like this.
- The disk is divided into tracks. Each track is further divided into sectors. The point to be noted here is that outer tracks are bigger in size than the inner tracks but they contain the same number of sectors and have equal storage capacity. This is because the storage density is high in sectors of the inner tracks whereas the bits are sparsely arranged in sectors of the outer tracks. Some space of every sector is used for formatting. So, the actual capacity of a sector is less than the given capacity.





- Read-Write(R-W) head moves over the rotating hard disk. It is this Read-Write head that performs all the read and writes operations on the disk and hence, the position of the R-W head is a major concern. To perform a read or write operation on a memory location, we need to place the R-W head over that position. Some important terms must be noted here:
  1. **Seek time –** The time taken by the R-W head to reach the desired track from its current position.
  2. **Rotational latency –** Time is taken by the sector to come under the R-W head.
  3. **Data transfer time –** Time is taken to transfer the required amount of data. It depends upon the rotational speed.
  4. **Controller time –** The processing time taken by the controller.
  5. **Average Access time –** seek time + Average Rotational latency + data transfer time + controller time.

For Example –

Consider a hard disk with:

- ***4 surfaces***
- ***64 tracks/surface***
- ***128 sectors/track***
- ***256 bytes/sector***

What is the capacity of the hard disk?

- ***Disk capacity = surfaces * tracks/surface * sectors/track * bytes/sector***
  *Disk capacity = 4 * 64 * 128 * 256*
  Disk capacity = 8 MB

The disk is rotating at 3600 RPM, what is the data transfer rate?

- *60 sec -> 3600 rotations*
  1 sec -> 60 rotations
  Data transfer rate = number of rotations per second * track capacity * number of surfaces (since 1 R-W head is used for each surface)
  *Data transfer rate = 60 * 128 * 256 * 4*
  Data transfer rate = 7.5 MB/sec

Note:

- Average Rotational latency is mostly 1/2*(Rotational latency).
- In questions, if the seek time and controller time are not mentioned, take them to be zero.
- If the amount of data to be transferred is not given, assume that no data is being transferred. Otherwise, calculate the time taken to transfer the given amount of data.
- The average rotational latency is taken when the current position of the R-W head is not given. Because the R-W may be already present at the desired position or it might take a whole rotation to get the desired sector under the R-W head. But, if the current position of the R-W head is given then the rotational latency must be calculated.

..Example . . .

The disk is rotating at 3600 RPM, what is the average access time?

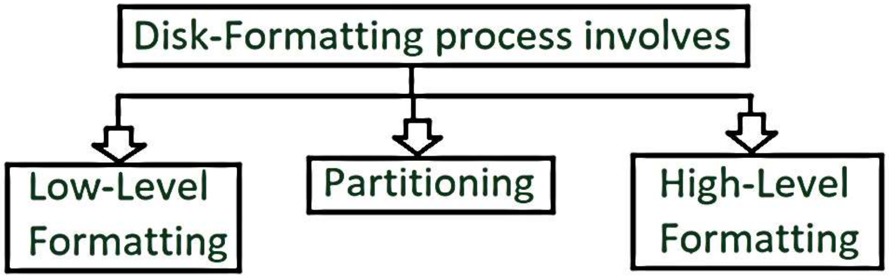
- *Since seek time, controller time and the amount of data to be transferred is not given, we consider all three terms as 0. Therefore, :*

  Average Access time = Average rotational delay
  Rotational latency => 60 sec -> 3600 rotations
  1 sec -> 60 rotations
  Rotational latency = (1/60) sec = 16.67 msec.
  Average Rotational latency = (16.67)/2
  = 8.33 msec.
  Average Access time = 8.33 msec.

4. Disk Formatting

- **Disk formatting** is a process to configure the data-storage devices such as hard-drive, floppy disk and flash drive when we are going to use them for the very first time or we can say initial usage. Disk formatting is usually required when new operating system is going to be used by the user. It is also done when there is space issue and we require additional space for the storage of more data in the drives. When we format the disk then the existing files within the disk is also erased.
- We can perform disk formatting on both magnetic platter hard-drives and solid-state drives.
- When we are going to use hard-drive for initial use it is going to search for virus. It can scan for virus and repair the bad sectors within the drive. Disk formatting has also the capability to erase the bad applications and various sophisticated viruses.
- As we know that disk formatting deletes data and removes all the programs installed with in the drive. So it can be done with caution. We must have the backup of all the data and applications which we require. No-doubt disk formatting requires time. But the frequent formatting of the disk decreases the life of the hard-drive.

Formatting process of disk:

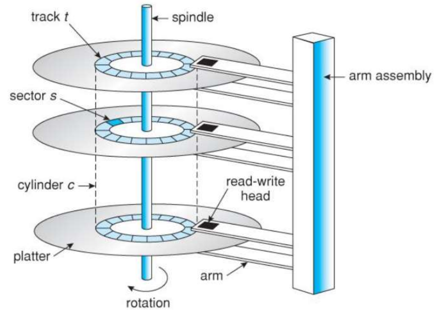


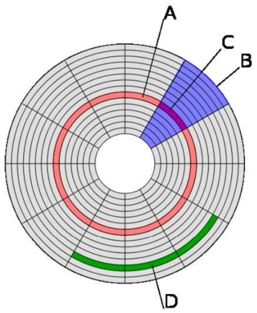
Disk-Formatting process involves

Low-Level Formatting

Partitioning

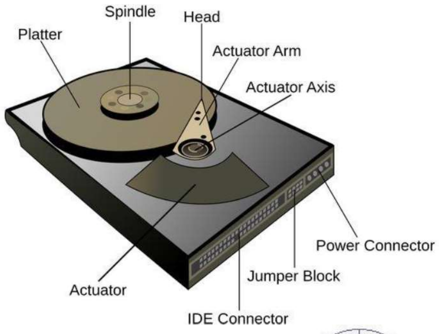
High-Level Formatting

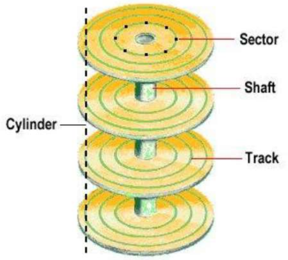


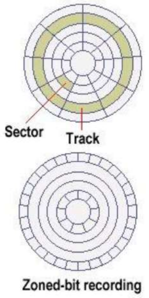


Hard Drive Structure:

A = track
B = sector
C = sector of a track
D = cluster







1. Low-level Formatting :

Low level formatting is a type of physical formatting. It is the process of marking of cylinders and tracks of the blank hard-disk. After this there is the division of tracks into sectors with the sector markers. Now-a-days low-level formatting is performed by the hard-disk manufactures themselves.

We have data in our hard-disk and when we perform low-level formatting in the presence of data in the hard-disk all the data have been erased and it is impossible to recover that data. Some users make such a format that they can avoid their privacy leakage. Otherwise low-level will cause damage to hard-disk shortens the service-life.

Therefore, this formatting is not suggested to users.

2. Partitioning :

As suggesting from the name, partitioning means divisions. Partitioning is the process of dividing the hard-disk into one or more regions. The regions are called as partitions.

It can be performed by the users and it will affect the disk performance.

3. High-level Formatting :

High-level formatting is the process of writing. Writing on a file system, cluster size, partition label, and so on for a newly created partition or volume. It is done to erase the hard-disk and again installing the operating system on the disk-drive.

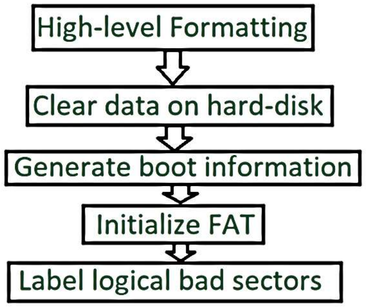


Figure – Steps of High-level Formatting

- Firstly High-level formatting clears the data on hard-disk, then it will generate boot information, then it will initialise FAT after this it will go for label logical bad sectors when partition has existed.
- Formatting done by the user is the high-level formatting.
- Generally, It does not harm the hard-disk.It can be done easily with the Administrator, Windows snap-in Disk Management tool, diskpart, etc.
- We can use such a format to fix some problems like errors in the file system, corrupted hard-drive and develop bad sectors.

Common File Systems:

- There are many different file systems in use today, each with its advantages and disadvantages. Here are some of the most common file systems −
- **NTFS** − NTFS (New Technology File System) is the default file system used by Windows operating systems since Windows XP. It offers features such as file compression, encryption, and permissions, making it suitable for use on local hard drives and network storage.
- **FAT32** − FAT32 (File Allocation Table 32) is an older file system that is still widely used on removable storage devices such as USB drives and memory cards. It is compatible with both Windows and Mac OS but has limitations such as a maximum file size of 4GB.
- **exFAT** − exFAT (Extended File Allocation Table) is a newer file system designed to overcome some of the limitations of FAT32. It supports larger file sizes and is more resistant to corruption, making it suitable for use on external storage devices.
- **HFS+** − HFS+ (Hierarchical File System Plus) is the default file system used by Mac OS X. It offers features such as file compression and encryption and is designed for use on local hard drives and external storage devices.

Choosing the Right File System

- When choosing a file system, it is important to consider the intended use of the storage device. Some factors to consider include compatibility with other devices and operating systems, support for large file sizes, and the need for advanced features such as file encryption or compression. It is also important to note that different file systems have different performance characteristics, so it may be necessary to benchmark the device with different file systems to determine which one provides the best performance for your specific use case.

Steps to Format a Disk(For Windows):

1. Connect the disk to your computer and ensure that it is recognized by the system.
2. Open File Explorer and locate the disk you want to format.
3. Right-click on the disk and select "Format" from the context menu.
4. In the "Format" dialog box, select the file system you want to use (e.g. NTFS, FAT32, exFAT) and choose a name for the volume.
5. If you want to perform a quick format, leave the "Quick Format" option checked. If you want to perform a full format, uncheck this option.
6. Click "Start" to begin the formatting process.
7. Wait for the formatting process to complete. This may take several minutes or longer, depending on the size of the disk and the formatting options you have chosen.
8. Once the formatting process is complete, the disk will be ready for use.

5. Disk Arm Scheduling Algorithms

- **Disk scheduling** is done by operating systems to schedule I/O requests arriving for the disk. Disk scheduling is also known as I/O Scheduling.
- **Importance of Disk Scheduling in Operating System**
  1. Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.
  2. Two or more requests may be far from each other so this can result in greater disk arm movement.
  3. Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner.

Recovery Options:

After formatting a disk, you may wonder if the data on it can be recovered. Here are some recovery options to consider −

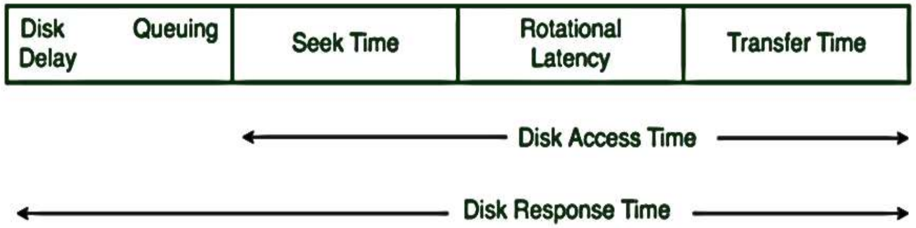
- **Data Recovery Software** − There are many data recovery software options available that can scan the formatted disk and attempt to recover lost data. Some popular options include Recuva, EaseUS Data Recovery Wizard, and Disk Drill.
- **Professional Data Recovery Services** − If the data is particularly important or the formatting process was done improperly, you may want to consider professional data recovery services. These companies specialize in recovering data from damaged or formatted disks.

There are number of algorithms in Disk Scheduling:

- *Disk Scheduling Algorithms*
  - *FCFS (First Come First Serve)*
  - *SSTF (Shortest Seek Time First)*
  - *SCAN (Elevator Algorithm)*
  - *C-SCAN (CIrcular SCAN)*
  - *LOOK*
  - *C-LOOK*
  - *RSS*
  - *LIFO (Last-In First-Out)*
  - *N-Step SCAN*
  - *F-SCAN, etc...*
- Among them, here **we discuss only** following three algorithms.
  1. First Come First Serve (FCFS)
  2. Shortest Seek Time First (SSTF)
  3. SCAN.

Important Terms Associated with Disk Scheduling:

1. **Seek Time:** Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or written. So the disk scheduling algorithm that gives a minimum average seek time is better.
2. **Rotational Latency:** Rotational Latency is the time taken by the desired sector of the disk to rotate into a position so that it can access the read/write heads. So the disk scheduling algorithm that gives minimum rotational latency is better.
3. **Transfer Time:** Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and the number of bytes to be transferred.
4. **Disk Access Time:**
   - *Disk Access Time = Seek Time + Rotational Latency + Transfer Time*
   - *Total Seek Time = Total head Movement * Seek Time*



Disk Queuing Delay | Seek Time | Rotational Latency | Transfer Time

Disk Access Time

Disk Response Time

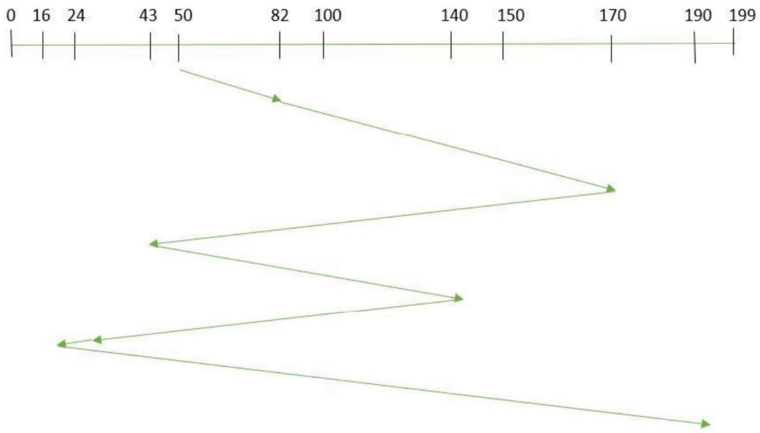
Disk Access Time and Disk Response Time

Important Terms Associated with Disk Scheduling:

5. **Disk Response Time:** Response Time is the average time spent by a request waiting to perform its I/O operation. The average *Response time* is the response time of all requests. *Variance Response Time* is the measure of how individual requests are serviced with respect to average response time. So the disk scheduling algorithm that gives minimum variance response time is better.

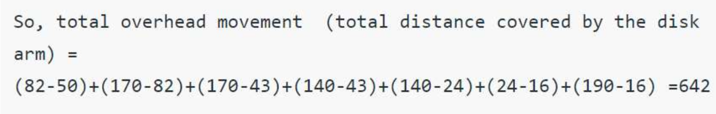
1. FCFS (First Come First Serve)

- FCFS is the simplest of all Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue. Let us understand this with the help of an example.



Example:

- Suppose the order of request is- (82,170,43,140,24,16,190)
  And current position of Read/Write head is: 50



```
So, total overhead movement  (total distance covered by the disk
arm) =
(82-50)+(170-82)+(170-43)+(140-43)+(140-24)+(24-16)+(190-16) =642
```

Advantages of FCFS

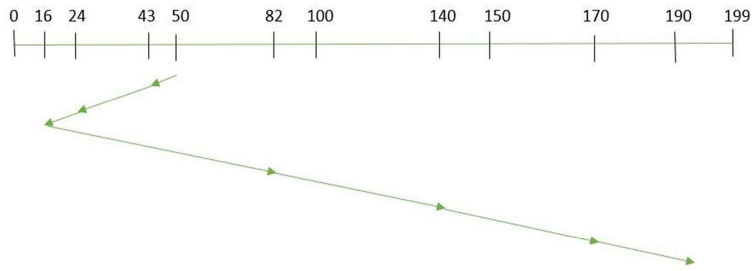
- Here are some of the advantages of First Come First Serve.
- Every request gets a fair chance
- No indefinite postponement

Disadvantages of FCFS

- Here are some of the disadvantages of First Come First Serve.
- Does not try to optimize seek time
- May not provide the best possible service

2. SSTF (Shortest Seek Time First)

- In SSTF (Shortest Seek Time First), requests having the shortest seek time are executed first. So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time.
- As a result, the request near the disk arm will get executed first. SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of the system. Let us understand this with the help of an example.



Example:

- Suppose the order of request is- (82,170,43,140,24,16,190)
  And current position of Read/Write head is: 50

```
total overhead movement  (total distance covered by the disk arm) =
(50-43)+(43-24)+(24-16)+(82-16)+(140-82)+(170-140)+(190-170) =208
```

Advantages of Shortest Seek Time First

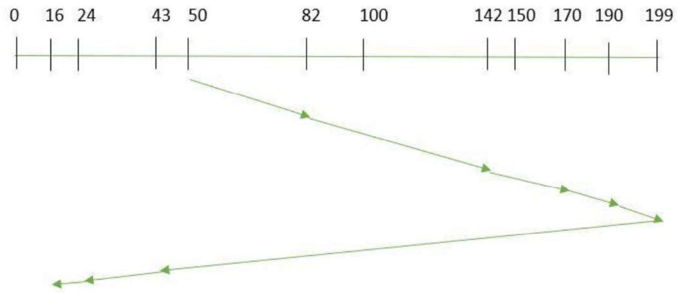
- Here are some of the advantages of Shortest Seek Time First.
- The average Response Time decreases
- Throughput increases

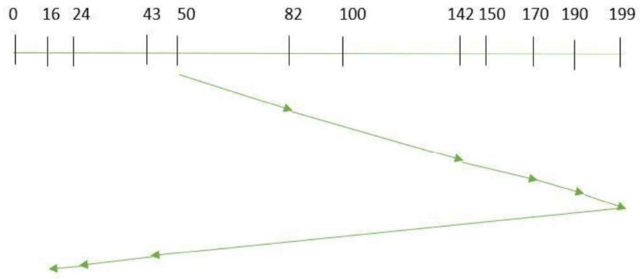
Disadvantages of Shortest Seek Time First

- Here are some of the disadvantages of Shortest Seek Time First.
- Overhead to calculate seek time in advance
- Can cause Starvation for a request if it has a higher seek time as compared to incoming requests
- The high variance of response time as SSTF favors only some requests

3. SCAN

- In the SCAN algorithm the disk arm moves in a particular direction and services the requests coming in its path and after reaching the end of the disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works as an elevator and is hence also known as an **elevator algorithm.** As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait. Example:





- Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"towards the larger value".**
- Therefore, the total overhead movement (total distance covered by the disk arm) is calculated as

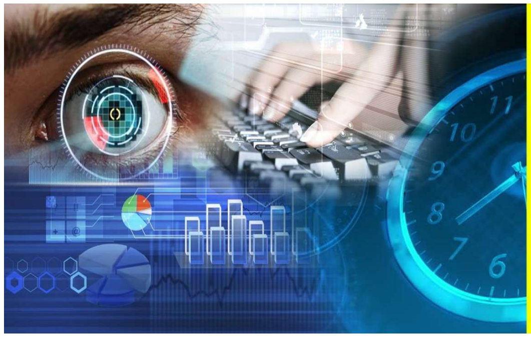
```
= (199-50) + (199-16) = 332
```

Advantages of SCAN Algorithm

- Here are some of the advantages of the SCAN Algorithm.
- High throughput
- Low variance of response time
- Average response time

Disadvantages of SCAN Algorithm

- Here are some of the disadvantages of the SCAN Algorithm.
- Long waiting time for requests for locations just visited by disk arm

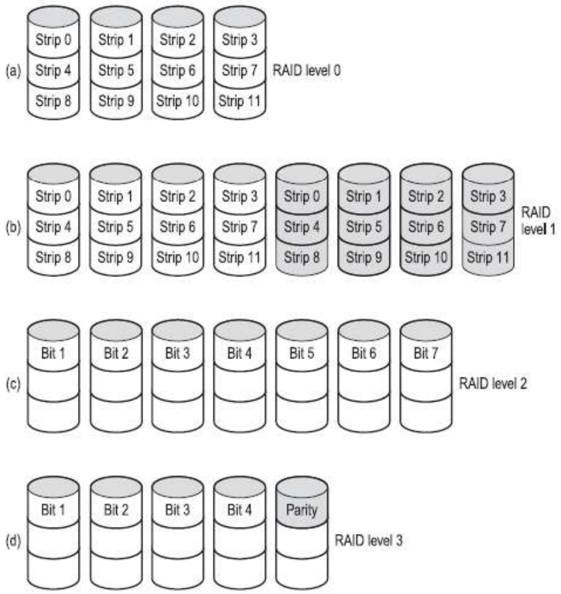


HNDIT 3052 Operating Systems

RAID Systems

(Input / Output management and Disk scheduling)

RAID levels 0 through 6. Backup and parity drives are shown shaded



RAID:

- In their 1988, Patterson suggested six specific disk organizations that could be used to improve disk performance, reliability, or both. These ideas were quickly adopted by industry and have led to a new class of I/O device called a **RAID**. Patterson defined RAID as **Redundant Array of Inexpensive Disks**, but industry redefined the I to be ‘‘Independent’’ rather than ‘‘Inexpensive’’.

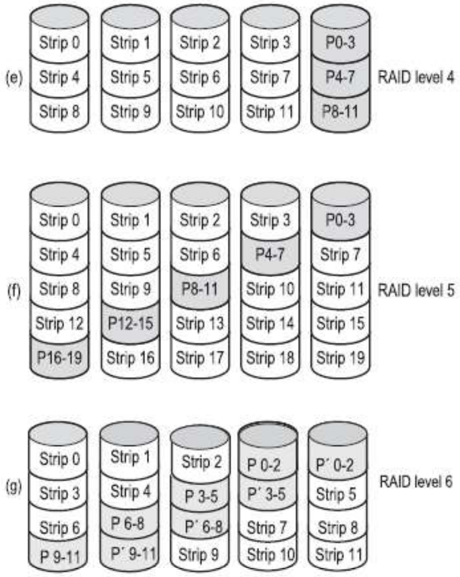
Why to use RAID?

- With the increasing demand in the storage and data world wide the prime concern for the organization is moving towards the security of their data. Now when I use the term security, here it does not means security from vulnerable attacks rather than from hard disk failures and any such relevant accidents which can lead to destruction of data. Now at those scenarios RAID plays it magic by giving you redundancy and an opportunity to get back all your data within a glimpse of time.

RAID Levels

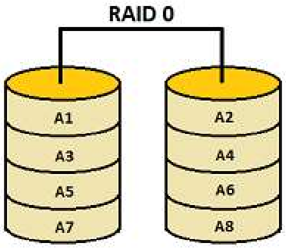
- Now with the moving generation and introduction of new technologies new RAID levels started coming into the picture with various improvisation giving an opportunity to organizations to select the required model of RAID as per their work requirement.

RAID levels 0 through 6. Backup and parity drives are shown shaded



- **RAID 0:** This level strips the data into multiple available drives equally giving a very high read and write performance but offering no fault tolerance or redundancy. This level does not provides any of the RAID factor and cannot be considered in an organization looking for redundancy instead it is preferred where high performance is required.

Pros	Cons
Data is stripped into multiple drives	No support for Data Redundancy
Disk space is fully utilized	No support for Fault Tolerance
Minimum 2 drives required	No error detection mechanism
High performance	Failure of either disk results in complete data loss in respective array



Calculation:
No. of Disk: 5
Size of each disk: 100GB
Usable Disk size: 500GB

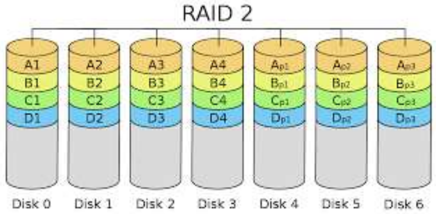
- **RAID 2**

- This level uses bit-level data stripping rather than block level. To be able to use RAID 2 make sure the disk selected has no self disk error checking mechanism as this level uses external Hamming code for error detection. This is one of the reason RAID is not in the existence in real IT world as most of the disks used these days come with self error detection. It uses an extra disk for storing all the parity information

Calculation:
Formula: n-1 where n is the no. of disk

If, No. of Disk: 3
Size of each disk: 100GB

Usable Disk size: 200GB

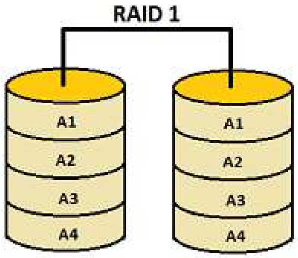


Pros	Cons
BIT level stripping with parity	It is used with drives with no built in error detection mechanism
One designated drive is used to store parity	These days all SCSI drives have error detection
Uses Hamming code for error detection	Additional drives required for error detection

- **RAID 1 :** This level performs mirroring of data in drive 1 to drive 2. It offers 100% redundancy as array will continue to work even if either disk fails. So organization looking for better redundancy can opt for this solution but again cost can become a factor.

Calculation:
No. of Disk: 2
Size of each disk: 100GB

Usable Disk size: 100GB

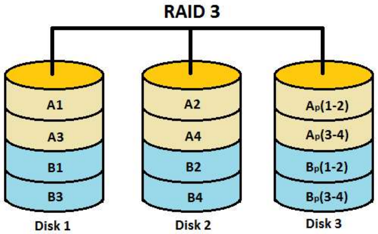


Pros	Cons
Performs mirroring of data i.e identical data from one drive is written to another drive for redundancy.	Expense is higher (1 extra drive required per drive for mirroring)
High read speed as either disk can be used if one disk is busy	Slow write performance as all drives has to be updated
Array will function even if any one of the drive fails	
Minimum 2 drives required	

RAID 3

This level uses byte level stripping along with parity. One dedicated drive is used to store the parity information and in case of any drive failure the parity is restored using this extra drive. But in case the parity drive crashes then the redundancy gets affected again so not much considered in organizations.

Calculation:
Formula: n-1 where n is the no. of disk
No. of Disk: 3
Size of each disk: 100GB
Usable Disk size: 200GB



Pros	Cons
BYTE level stripping with parity	Additional drives required for parity
One designated drive is used to store parity	No redundancy in case parity drive crashes
Data is regenerated using parity drive	Slow performance for operating on small sized files
Data is accessed parallel	
High data transfer rates (for large sized files)	
Minimum 3 drives required	

RAID 4: This level is very much similar to RAID 3 apart from the feature where RAID 4 uses block level stripping rather than byte level.

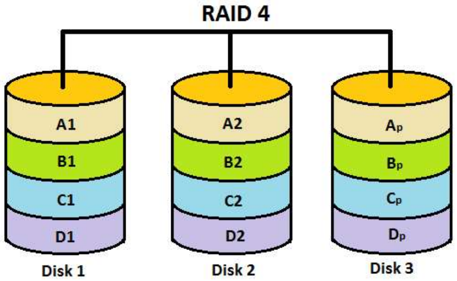
Calculation:

Formula: n-1 where n is the no. of disk

No. of Disk: 3

Size of each disk: 100GB

Usable Disk size: 200GB



Pros	Cons
BLOCK level stripping along with dedicated parity	Since only 1 block is accessed at a time so performance degrades
One designated drive is used to store parity	Additional drives required for parity
Data is accessed independently	Write operation becomes slow as every time a parity has to be entered
Minimum 3 drives required	
High read performance since data is accessed independently.	

RAID 6: This level is an enhanced version of RAID 5 adding extra benefit of dual parity. This level uses block level stripping with DUAL distributed parity. So now you can get extra redundancy. Imagine you are using RAID 5 and 1 of your disk fails so you need to hurry to replace the failed disk because if simultaneously another disk fails then you won't be able to recover any of the data so for those situations RAID 6 plays its part where you can survive 2 concurrent disk failures before you run out of options.

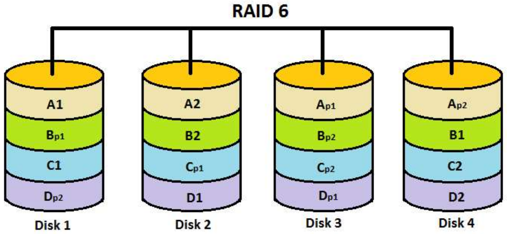
Calculation:

Formula: n-2 where n is the no. of disk

No. of Disk: 4

Size of each disk: 100GB

Usable Disk size: 200GB



Pros	Cons
Block level stripping with DUAL distributed parity	Cost Expense can become a factor
2 parity blocks are created	Writing data takes longer time due to dual parity
Can survive concurrent 2 drive failures in an array	
Extra Fault Tolerance and Redundancy	
Minimum 4 drives required	

RAID 5: It uses block level stripping and with this level distributed parity concept came into the picture leaving behind the traditional dedicated parity as used in RAID 3 and RAID 5. Parity information is written to a different disk in the array for each stripe. In case of single disk failure data can be recovered with the help of distributed parity without affecting the operation and other read write operatior

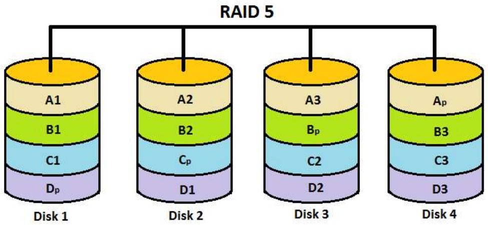
Calculation:

Formula: n-1 where n is the no. of disk

No. of Disk: 4

Size of each disk: 100GB

Usable Disk size: 300GB



Pros	Cons
Block level stripping with DISTRIBUTED parity	In case of disk failure recovery may take longer time as parity has to be calculated from all available drives
Parity is distributed across the disks in an array	Cannot survive concurrent drive failures
High Performance	
Cost effective	
Minimum 3 drives required	

RAID 0+1: This level uses RAID 0 and RAID 1 for providing redundancy. Stripping of data is performed before Mirroring. In this level the overall capacity of usable drives is reduced as compared to other RAID levels. You can sustain more than one drive failure as long as they are not in the same mirrored set.

NOTE: The no. of drives to be created should always be in the multiple of 2

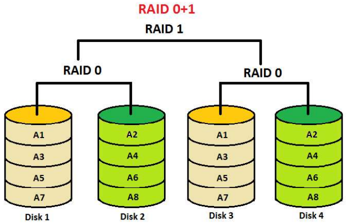
Calculation:

Formula: n/2 * size of disk (where n is the no. of disk)

No. of Disk: 8

Size of each disk: 100GB

Usable Disk size: 400GB

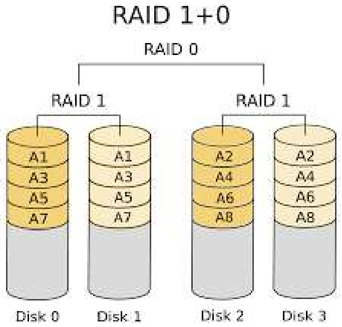


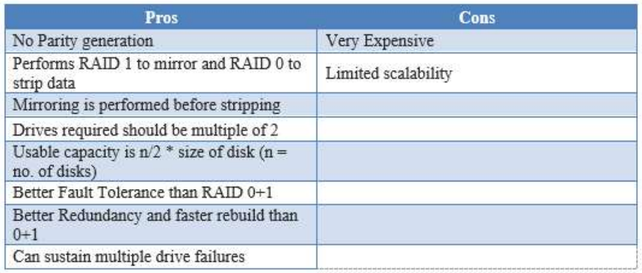
Pros	Cons
No parity generation	Costly as extra drive is required for each drive
Performs RAID 0 to strip data and RAID 1 to mirror	100% disk capacity is not utilized as half is used for mirroring
Stripping is performed before Mirroring	Very limited scalability
Usable capacity is n/2 * size of disk (n = no. of disks)	
Drives required should be multiple of 2	
High Performance as data is stripped	

RAID 1+0 (RAID 10): This level performs Mirroring of data prior stripping which makes it much more efficient and redundant as compared to RAID 0+1. This level can survive multiple simultaneous drive failures. This can be used in organizations where high performance and security are required. In terms of fault Tolerance and rebuild performance it is better than RAID 0+1.

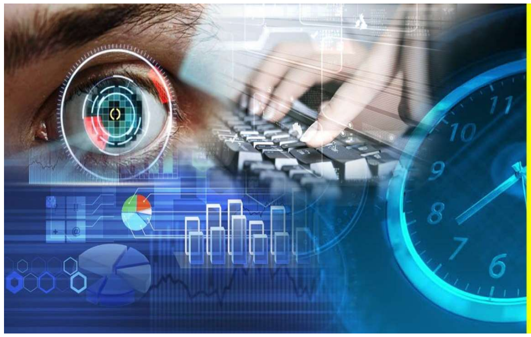
Calculation:

Formula: n/2 * size of disk (where n is the no. of disk)
No. of Disk: 8
Size of each disk: 100GB
Usable Disk size: 400GB





Pros	Cons
No Parity generation	Very Expensive
Performs RAID 1 to mirror and RAID 0 to strip data	Limited scalability
Mirroring is performed before stripping	
Drives required should be multiple of 2	
Usable capacity is n/2 * size of disk (n = no. of disks)	
Better Fault Tolerance than RAID 0+1	
Better Redundancy and faster rebuild than 0+1	
Can sustain multiple drive failures	



HNDIT 3052 Operating Systems

Unix, Linux, and Android

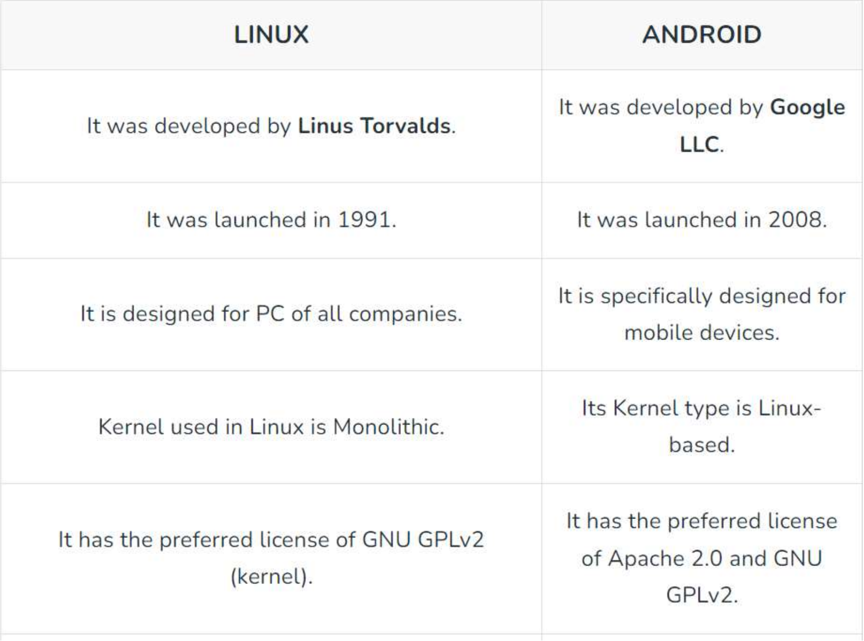
- **Difference between Unix and Linux**
- **Difference between Linux and Android**
- (Case Study) Working with Linux Distros
- challenges of designing operating systems for different platforms;
- Practice Installation and basic configurations of Unix based systems

Difference between Unix and Linux

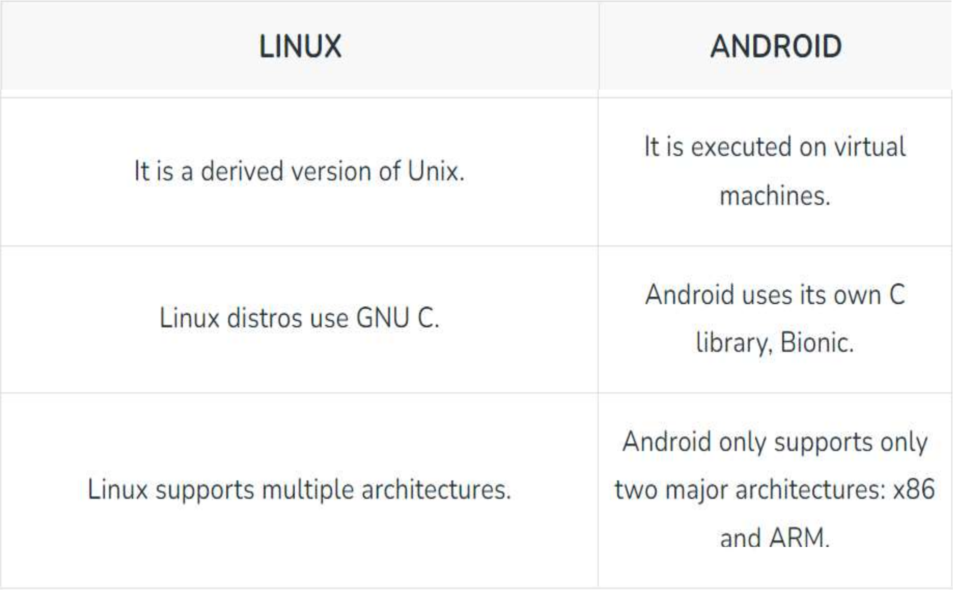
The Linux:

- Linux is a group of open-source Unix-like operating systems which was developed by Linus Torvalds. It is a package of Linux distribution.
- Some of the most used Linux distributions are Debian, Fedora, and Ubuntu. It was basically written in C language and assembly language.
- The kernel used in Linux is a Monolithic kernel. The target systems of Linux distributions are cloud computing, embedded systems, mobile devices, personal computers, servers, mainframe computers, and supercomputers. The first version of Linux was launched in 1991.
- The most recent version of Linux for personal computers is 5.6 (kernel).

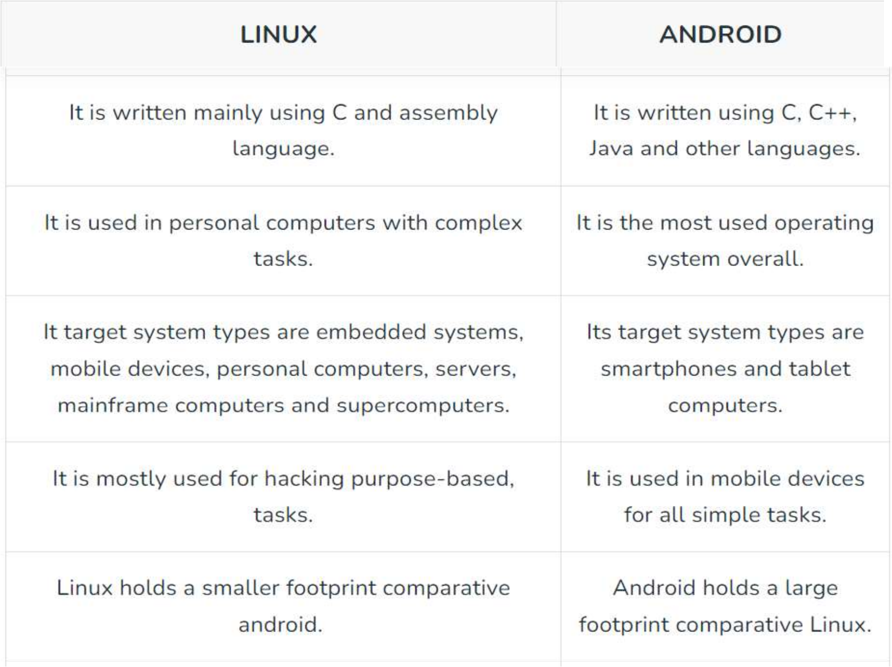
- Android is a mobile operating system that is provided by Google. It is based on the modified version of the Linux kernel and other open-source software.
- It is specifically designed for touchscreen mobile devices like smartphones and tablets.
- It was developed using C, Java, C++, and other languages.
- The first version of Android was launched by Google in 2008. The latest stable version of Android is Android 10. It is provided totally free of cost.
- It is the most used operating system overall.



LINUX	ANDROID
It was developed by **Linus Torvalds**.	It was developed by **Google LLC**.
It was launched in 1991.	It was launched in 2008.
It is designed for PC of all companies.	It is specifically designed for mobile devices.
Kernel used in Linux is Monolithic.	Its Kernel type is Linux-based.
It has the preferred license of GNU GPLv2 (kernel).	It has the preferred license of Apache 2.0 and GNU GPLv2.



LINUX	ANDROID
It is a derived version of Unix.	It is executed on virtual machines.
Linux distros use GNU C.	Android uses its own C library, Bionic.
Linux supports multiple architectures.	Android only supports only two major architectures: x86 and ARM.



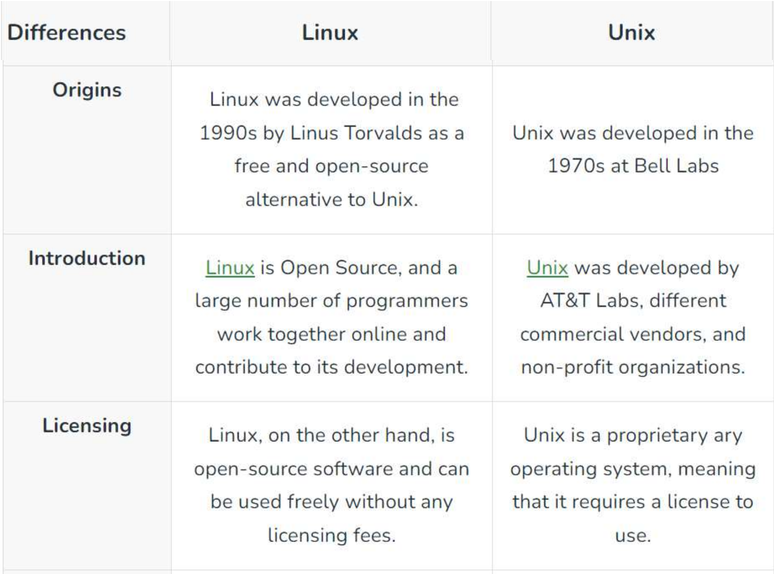
LINUX	ANDROID
It is written mainly using C and assembly language.	It is written using C, C++, Java and other languages.
It is used in personal computers with complex tasks.	It is the most used operating system overall.
It target system types are embedded systems, mobile devices, personal computers, servers, mainframe computers and supercomputers.	Its target system types are smartphones and tablet computers.
It is mostly used for hacking purpose-based, tasks.	It is used in mobile devices for all simple tasks.
Linux holds a smaller footprint comparative android.	Android holds a large footprint comparative Linux.

Difference between Unix and Linux

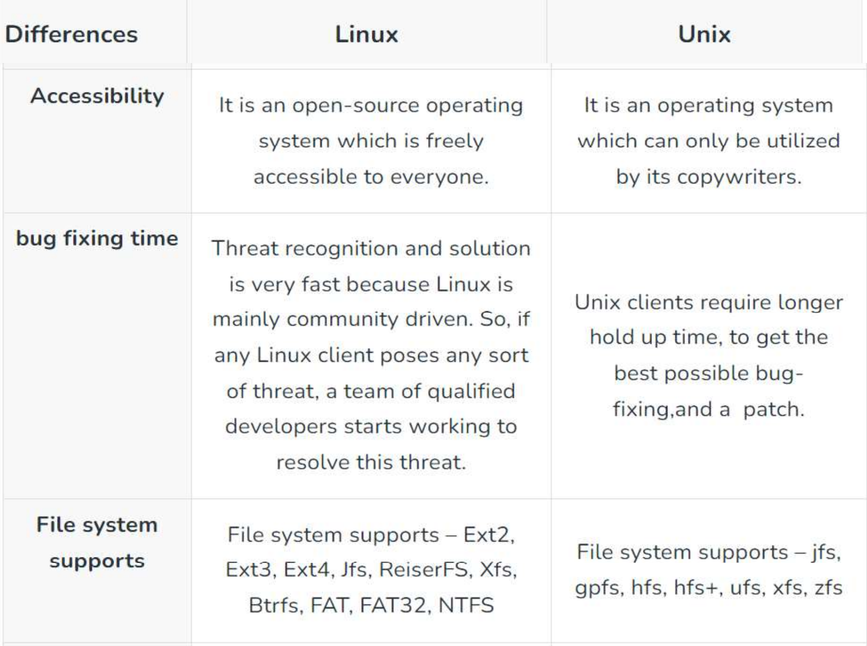
- **Linux** is an operating system that was developed by Linus Torvalds in 1991. The name "Linux" originates from the Linux kernel. It is an open-source software that is completely free to use. It is used for computer hardware and software, game development, mainframes, etc. It can run various client programs.
- **Unix** is a portable, multi-tasking, bug-fixing, multi-user operating system developed by AT&T. It started as a one-man venture under the initiative of Ken Thompson of Bell Labs. It proceeded to turn out to become the most widely used operating system. It is used in web servers, workstations, and PCs. Many business applications are accessible on it.

Linux and Unix are both operating systems that are commonly used in enterprise and server environments.

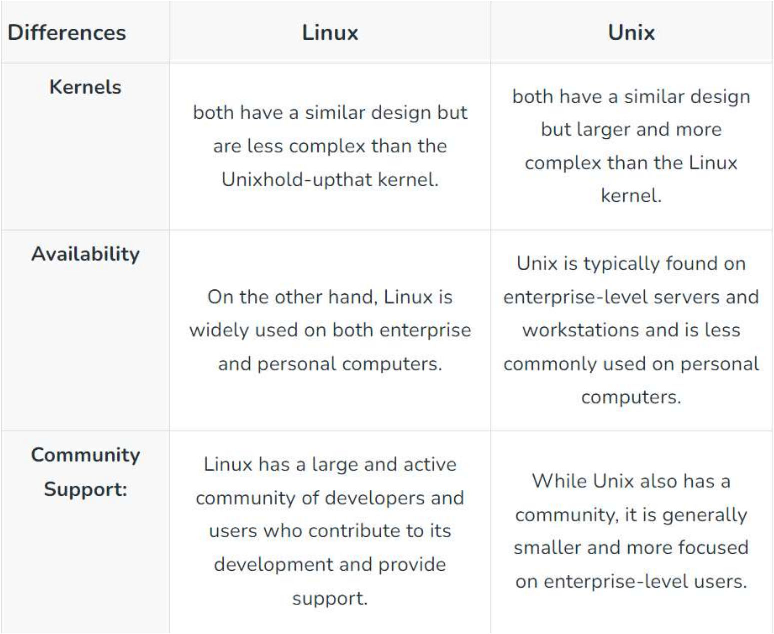
What are the differences between Linux and Unix?



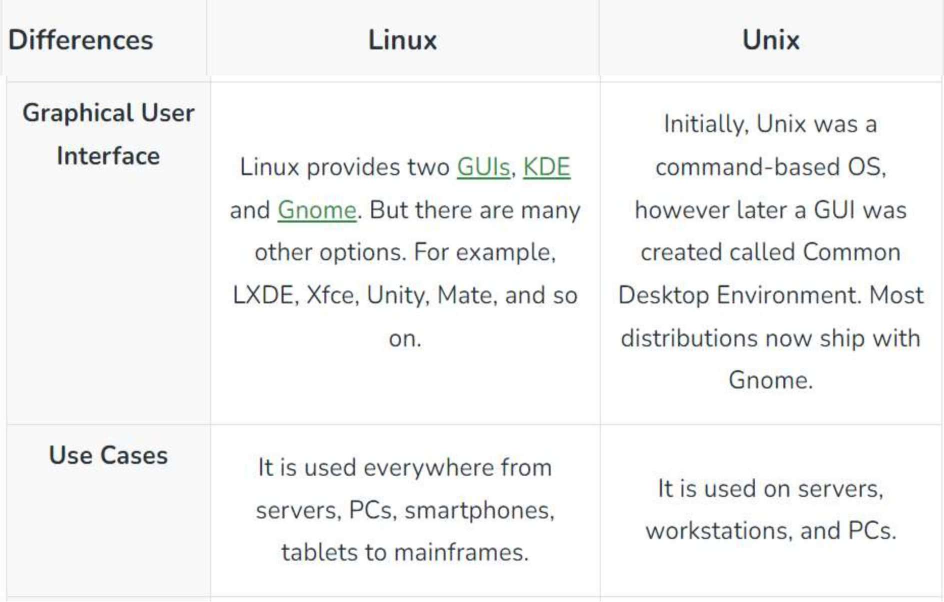
Differences	Linux	Unix
Origins	Linux was developed in the 1990s by Linus Torvalds as a free and open-source alternative to Unix.	Unix was developed in the 1970s at Bell Labs
Introduction	Linux is Open Source, and a large number of programmers work together online and contribute to its development.	Unix was developed by AT&T Labs, different commercial vendors, and non-profit organizations.
Licensing	Linux, on the other hand, is open-source software and can be used freely without any licensing fees.	Unix is a proprietary ary operating system, meaning that it requires a license to use.



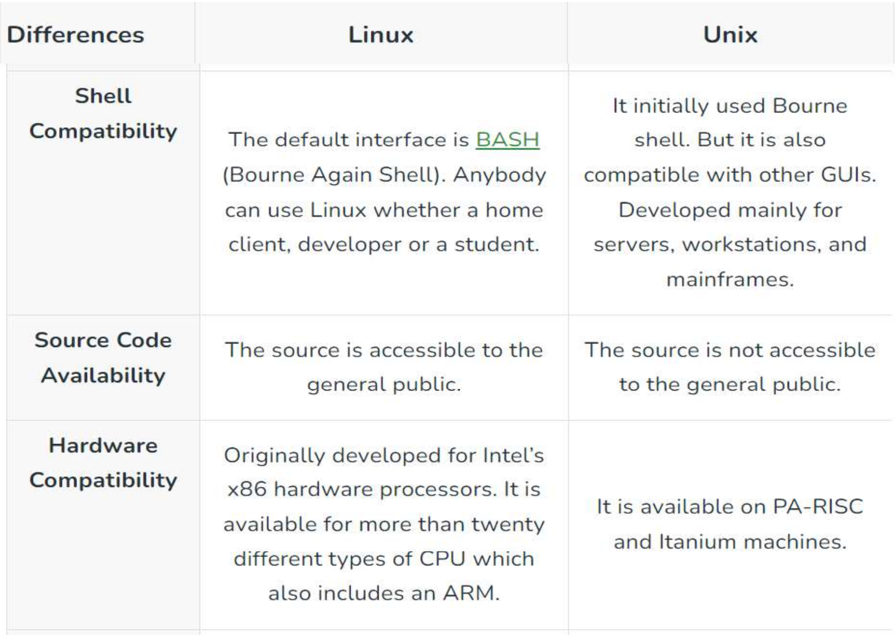
Differences	Linux	Unix
Accessibility	It is an open-source operating system which is freely accessible to everyone.	It is an operating system which can only be utilized by its copywriters.
bug fixing time	Threat recognition and solution is very fast because Linux is mainly community driven. So, if any Linux client poses any sort of threat, a team of qualified developers starts working to resolve this threat.	Unix clients require longer hold up time, to get the best possible bug-fixing,and a patch.
File system supports	File system supports – Ext2, Ext3, Ext4, Jfs, ReiserFS, Xfs, Btrfs, FAT, FAT32, NTFS	File system supports – jfs, gpfs, hfs, hfs+, ufs, xfs, zfs



Differences	Linux	Unix
Kernels	both have a similar design but are less complex than the Unixhold-upthat kernel.	both have a similar design but larger and more complex than the Linux kernel.
Availability	On the other hand, Linux is widely used on both enterprise and personal computers.	Unix is typically found on enterprise-level servers and workstations and is less commonly used on personal computers.
Community Support:	Linux has a large and active community of developers and users who contribute to its development and provide support.	While Unix also has a community, it is generally smaller and more focused on enterprise-level users.



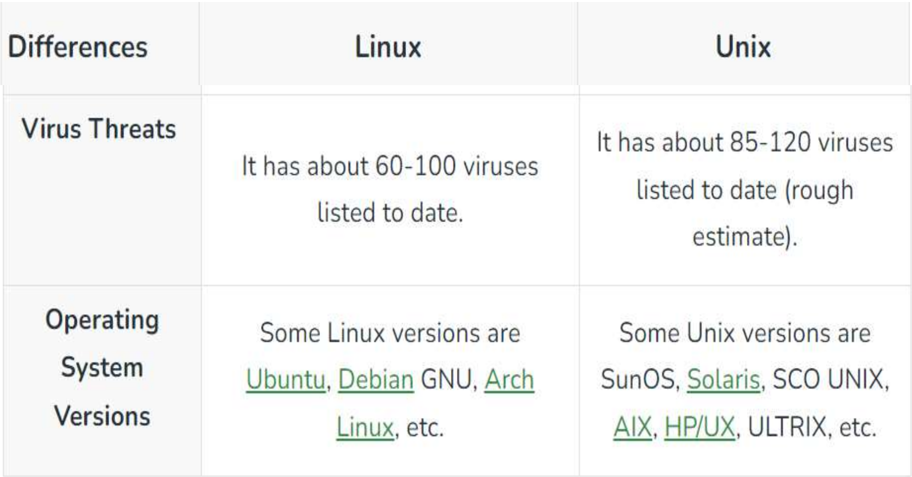
Differences	Linux	Unix
Graphical User Interface	Linux provides two GUIs, KDE and Gnome. But there are many other options. For example, LXDE, Xfce, Unity, Mate, and so on.	Initially, Unix was a command-based OS, however later a GUI was created called Common Desktop Environment. Most distributions now ship with Gnome.
Use Cases	It is used everywhere from servers, PCs, smartphones, tablets to mainframes.	It is used on servers, workstations, and PCs.



Differences	Linux	Unix
Shell Compatibility	The default interface is BASH (Bourne Again Shell). Anybody can use Linux whether a home client, developer or a student.	It initially used Bourne shell. But it is also compatible with other GUIs. Developed mainly for servers, workstations, and mainframes.
Source Code Availability	The source is accessible to the general public.	The source is not accessible to the general public.
Hardware Compatibility	Originally developed for Intel's x86 hardware processors. It is available for more than twenty different types of CPU which also includes an ARM.	It is available on PA-RISC and Itanium machines.

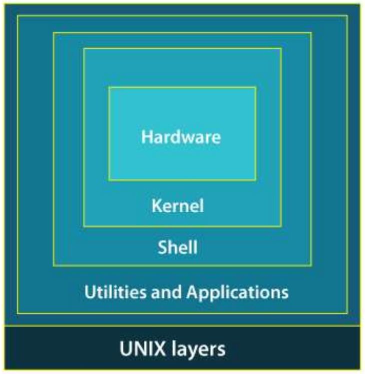
Features of UNIX Operating System:

- **Multitasking:**
- **Multi-user:**
- **Portability:**
- **File Security and Protection:**
- **Command Structure:**
- **Communication:**
- **Open Source:**
- **Accounting:**
- **UNIX Tools and Utilities:**

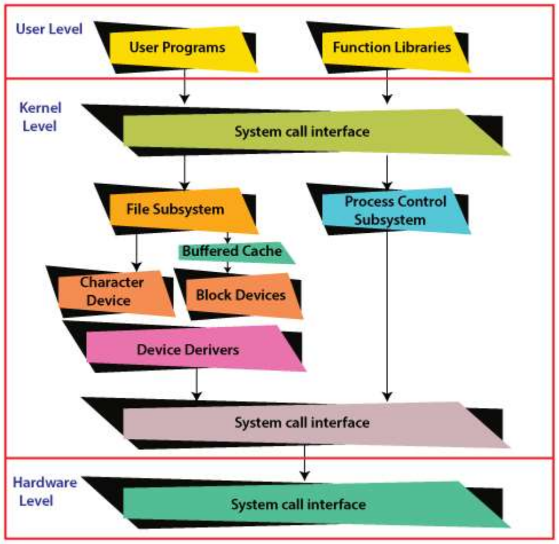


Differences	Linux	Unix
Virus Threats	It has about 60-100 viruses listed to date.	It has about 85-120 viruses listed to date (rough estimate).
Operating System Versions	Some Linux versions are Ubuntu, Debian GNU, Arch Linux, etc.	Some Unix versions are SunOS, Solaris, SCO UNIX, AIX, HP/UX, ULTRIX, etc.

1. The structure of Unix OS Layers, 2. Kernel Architecture



1. The structure of Unix OS Layers



Kernel Architecture

Linux Distributions (Distros)

Introduction to Linux Distribution

- Other operating systems like Microsoft combine each bit of codes internally and release it as a single package. You have to choose from one of the version they offer.
- But Linux is different from them. Different parts of Linux are developed by different organizations.
- Different parts include kernel, shell utilities, X server, system environment, graphical programs, etc. If you want you can access the codes of all these parts and assemble them yourself. But its not an easy task seeking a lot of time and all the parts has to be assembled correctly in order to work properly.
- From here on distribution (also called as distros) comes into the picture. They assemble all these parts for us and give us a compiled operating system of Linux to install and use.
- A Linux distribution is an OS made through a software collection that contains the Linux kernel and a package management system often.
- Usually, Linux users obtain their OS by downloading a Linux distribution, available for a range of systems from embedded devices (e.g., ***OpenWrt***) to robust supercomputers (e.g., Rocks Cluster Distribution).
- A Linux distribution **is composed** of a **Linux kernel**, **GNU libraries and tools**, **other software**, **a window system**, **documentation**, **a desktop environment**, and a **window manager**.
- Almost every added software is open-source and free and becomes available both as in source code and compiled binary form, permitting changes to the actual software.
- Optionally, Linux distributions add a few proprietary software that might not be available in the source code form, like binary blocks needed for a few device drivers.

Linux Distributions List:

(1) Ubuntu:

It came into existence in 2004 by Canonical and quickly became popular. Canonical wants Ubuntu to be used as easy graphical Linux desktop without the use of command line. It is the most well known Linux distribution. Ubuntu is a next version of Debian and easy to use for newbies. It comes with a lots of pre-installed apps and easy to use repositories libraries.

Earlier, Ubuntu uses GNOME2 desktop environment but now it has developed its own unity desktop environment. It releases every six months and currently working to expand to run on tablets and smartphones.

(2) Linux Mint:

Mint is based on Ubuntu and uses its repository software so some packages are common in both.

Earlier it was an alternative of Ubuntu because media codecs and proprietary software are included in mint but was absent in Ubuntu. But now it has its own popularity and it uses cinnamon and mate desktop instead of Ubuntu's unity desktop environment.

Trends and types Linux distributions might be:

- Non-commercial or commercial
- Developed for home users, power users, or enterprise users
- Supported on **two** or **more types** of **platform** or hardware-specific, even to the certification extension via platform vendor
- Developed for embedded, desktop, or server devices
- Highly specialized or general purpose toward particular machine functionalities (e.g., computer clusters, network routers, and firewalls)
- Targeted at particular user groups, e.g., by **language internationalization** and **localization** or by including several **scientific computing** and **music production** packages
- Primarily, built for comprehensiveness, portability, usability, or security
  Rolling release or standard release.

(3) Debian: Debian has its existence since 1993 and releases its versions much slowly then Ubuntu and mint. This makes it one of the most stable Linux distributor.

Ubuntu is based on Debian and was founded to improve the core bits of Debian more quickly and make it more user friendly. Every release name of Debian is based on the name of the movie Toy Story.

(4) Red Hat Enterprise / CentOS: Red hat is a commercial Linux distributor. There products are red hat enterprise Linux (RHEL) and Fedora which are freely available. RHEL is well tested before release and supported till seven years after the release, whereas, fedora provides faster update and without any support.

Red hat uses trademark law to prevent their software from being redistributed. CentOS is a community project that uses red hat enterprise Linux code but removes all its trademark and make it freely available. In other words, it is a free version of RHEL and provide a stable platform for a long time.

(5) Fedora: It is a project that mainly focuses on free software and provides latest version of software. It doesn't make its own desktop environment but used 'upstream' software. By default it has GNOME3 desktop environment. It is less stable but provides the latest stuff.

Table 1: Evolution of Linux

S.No.	Year	Event/Release	Version
1	1991	UniX(HP-UX)	8.0
2	1992	Hewlett Packard	9.0
3	1993	NetBSD and FreeBSD	0.8, 1.0
4	1994	Red Hat Linux, Caldera, Ransom Love and NetBSD1.0	—
5	1995	FreeBSD and HP UX	2.0, 10.0
6	1996	K Desktop Environment	—
7	1997	IIP-UX	11.0
8	1998	IRIXSun Solaris OS Free BSD	6.573.0
9	2000	Caldera Systems with SCO server division announced	
10	2001	LinuxMicrosoft filed a trademark suit against Lindows.com	2.4
11	2004	Lindows name was changed to Linspire First release of Ubuntu	—
12	2005	openSUSE Project	—
13	2006	Red Hat	—
14	2007	Dell started distributing laptops with Ubuntu pre-installed	
16	2011	Linux kernel	3.0
17	2013	Googles Linux based Android	—

Linux has several distributions. Some popular and dominant LINUX distributions for Desktops include:

- Linux Mint
- Ubuntu
- OpenSUSE
- Mageia
- Manjaro
- Fedora
- Arch Linux
- Debian
- Kali Linux
- Elementary OS

Some of the popular Linux distributions, in the Servers category, include:

- Red Hat Enterprise Linux (RHEL)
- CentOS
- Ubuntu Server
- SUSE Enterprise Linux

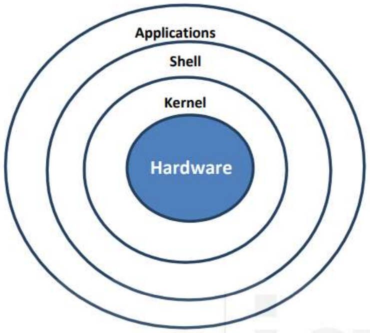
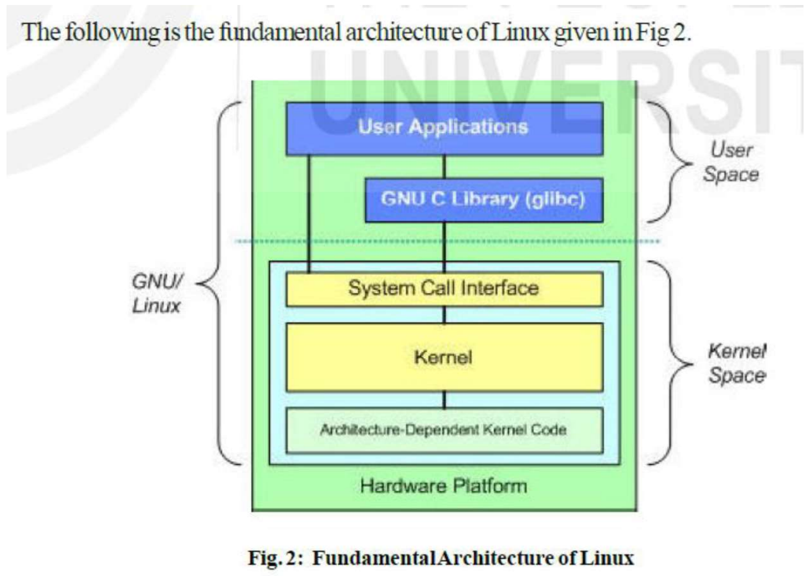


Fig 1: Linux Architecture

Kernel Architecture of Linux

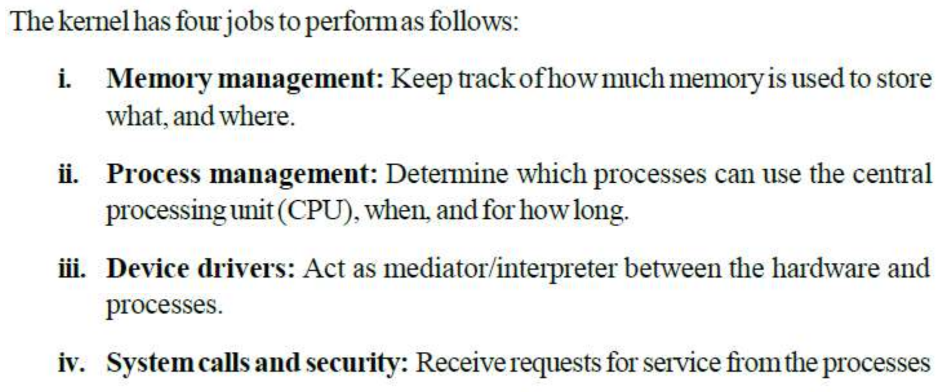
- Kernel is a small and special code which is the core component of Linux OS and directly interacts with hardware.
- It is the intermediate level between software and hardware which provides low level service to user mode's components.
- It is developed in C language. Moreover, it has different blocks which manage various operations. Kernel runs a number of processes concurrently and manages various resources.
- It is viewed as a resource manager when several programs run concurrently on a system. In this case, the kernel is an instance that shares available resources like CPU time, disk space, network connections etc.

FUNDAMENTAL ARCHITECTURE OF LINUX



The following is the fundamental architecture of Linux given in Fig 2.

Fig. 2: Fundamental Architecture of Linux



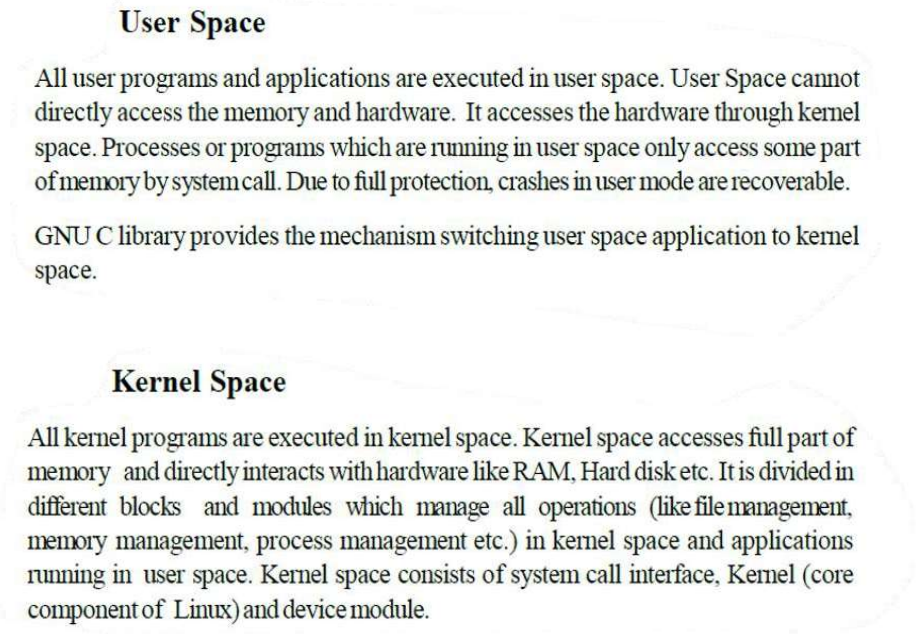
The kernel has four jobs to perform as follows:

i. **Memory management:** Keep track of how much memory is used to store what, and where.

ii. **Process management:** Determine which processes can use the central processing unit (CPU), when, and for how long.

iii. **Device drivers:** Act as mediator/interpreter between the hardware and processes.

iv. **System calls and security:** Receive requests for service from the processes



User Space

All user programs and applications are executed in user space. User Space cannot directly access the memory and hardware. It accesses the hardware through kernel space. Processes or programs which are running in user space only access some part of memory by system call. Due to full protection, crashes in user mode are recoverable.

GNU C library provides the mechanism switching user space application to kernel space.

Kernel Space

All kernel programs are executed in kernel space. Kernel space accesses full part of memory and directly interacts with hardware like RAM, Hard disk etc. It is divided in different blocks and modules which manage all operations (like file management, memory management, process management etc.) in kernel space and applications running in user space. Kernel space consists of system call interface, Kernel (core component of Linux) and device module.

Linux Niche distributions

Some other distributions require specific niches, including:

1. **Routers:** e.g., targeted by OpenWrt (the Tiny embedded router distribution)
2. **Internet of things:** e.g., targeted by Microsoft's Azure Sphere and Ubuntu Core
3. **Home theatre PCs:** e.g., targeted by Mythbuntu, Kodi (formerly XBMC), and KnoppMyth
4. **Specific platforms:** e.g., the Raspberry Pi platform is targeted by Raspberry Pi OS
5. **Education:** some examples are Karoshi and Edubuntu, and server systems are PCLinuxOS-based
6. **Scientific workstations and computer servers:** e.g., aimed by Scientific Linux
7. **Penetration testing, digital forensics, and computer security:** some examples are Parrot Security OS and Kali Linux
8. **Anonymity and privacy:** e.g., targeted by FreedomBox, Qubes, Whonix, or Tails
9. **Gaming:** e.g., SteamOS
10. **Offline use:** e.g., Endless OS

Practice Linux

1. Linux commands
   i. Linux file and directory management commands.
      1. cd
      2. ls
      3. touch
      4. cat
      5. mv

   ii. Linux security
      1. **Permissions**
      2. **Use A Firewall To Protect The Network**
      3. **Protecting wireless networks**
      4. **Networking protocols**
      5. **Using the Blocking Technique**
      6. **Anti-Virus**
      7. **Software Updates**
      8. **Security Testing**

References

1. https://www.geeksforgeeks.org/difference-between-linux-and-android/
2. https://www.geeksforgeeks.org/linux-vs-unix/
3. https://egyankosh.ac.in/bitstream/123456789/72530/3/Unit-2.pdf
4. https://www.linkedin.com/advice/3/how-do-you-navigate-challenges-operating-system
5. https://www.codingninjas.com/studio/library/

Linux Security

Introduction

Linux has been in great demand due to its fancy features and ease of access. But how does it manage to provide security? To know about the security aspect of Linux, keep reading!

Linux users are less prone to viruses than other operating systems, but many security issues still need to be taken care of.

What Are The Various Security Requirements?

We must fulfill some security requirements due to laws and regulations. Other security aspects are for the user and the safety of data.
We need to consider the following factors-

- **Authorization** - Only allow limited access
- **Authenticity** - Verifying the identity
- **Privacy and Integrity**- Ensure personal information and data are not compromised.
- **Availability** - We need to ensure that the system can perform its required function.

The next question may arise: why do we need to impose so much security? Who are we protecting our systems from? How do they endanger the integrity of your system?
Let us look at the answers to these questions.

Threats And Vulnerabilities

Threats

- **Theft Of Data**- Our computers have private information that we should protect. If this data goes into unsafe hands, it can lead to damages and losses.
- **Manipulation Of Data**- If someone not supposed to access the data gets it somehow, they can modify its contents. This is a more considerable risk as the authenticity of information is lost.
- **Denial Of Service**-Denial of Service (DoS) attacks are where the attacker disables or makes unusable the services provided by the system. These types of attacks happened with companies like Yahoo in the 2000s. These are complex attacks and need to be tackled.
- **Direct Access To Computer**- It is a common practice that we can use your computer to cause attacks by logging into other user accounts. We can use your machine to cause damage, and this situation becomes highly disgraceful.

Security Measures

Security Measures

The following **security measures** are the most **powerful** ones. Let us look at each one of them.

Permissions

The conventional authentication method uses a username and password. The username is not-so-hidden information, as everyone can see, Although the password is encrypted. It is only known to the user. The algorithm employed for encryption is one-way and rechecked every time a user enters the password.
In UNIX, the details of the passwords are present in the/etc/passwd file. This is prone to harm. Linux OS prevents this issue by using a shadow password file with restricted permission to read.

Use A Firewall To Protect The Network

Once your system is connected to the network, it becomes more vulnerable. We need to make sure that the transmitted data is safe. Se can do this by separating the network of computers and other networks. We can achieve this with the help of a firewall. A firewall determines whether traffic is allowed to pass or not. Firewalls are protection devices that act as a borderline between trusted and unsafe networks.

Protecting wireless networks

Using a firewall is a wise decision, but it has disadvantages. Network signals and firewalls can be messed up. We can secure the network by using WPA wireless network encryption. We can also apply tunneling on network traffic by using VPNs.

Networking protocols

We should insist on safer protocols like SSH, which transmits the data after encrypting it. The encrypted information is coded data and is not understandable without proper decryption. Outsiders can access the shared data if we continue using standard unsecured protocols like File Transfer Protocol and Telnet. This data is tampered with because it is unencrypted. We should employ more physical security to make data transmission safe and reliable.

Using the Blocking Technique

Another method to protect your local machine from the external network is to utilize specific blocking techniques. We can block clear network access and inbound links by using personal firewalls and restricting access using configuration. We can also use other software like Tcpwrappers to employ restrictions and save the data.

Anti-Virus

Although Linux is a fancy operating system that does not require explicit Anti Virus software, it can still be a carrier and affect other operating systems which are more vulnerable. It is, therefore, a healthy practice to occasionally run scans on your system to ensure no virus. Several anti-virus packages are available to do this job.

Software Updates

No software is safe from bugs. Whenever we encounter a bug at the security level, we need to find a solution for it. We can install a software update checker to achieve this in Linux distributions. This is included in Linux. The system automatically performs checks on data getting installed from unknown sources and fixes the bug wherever found. It also manually updates itself if needed.

Security Testing

After completing the system and the network operations, we should test to see if there are any more vulnerabilities to tackle. We can achieve this by using a network port scanning or dedicated security software tester. It tests the system and network for all possible vulnerabilities. This practice of software testing for access is penetration testing. There are many open-source tools available on the internet to support security testing. One example of such a feature is the open-source network scanner, Nmap. It is used to check the vulnerabilities of the network. We should remember that we can only apply these testing methods to authorized systems. It is a criminal act to run these checks on unauthorized ones.

Frequently Asked Questions

Mention the various requirements of security.

Authorization, Authenticity, Privacy, Integrity, and Availability are security requirements.

Why should we run Anti-virus scans on Linux OS?

Although Linux is a fancy operating system that does not require explicit Anti Virus software, it can still be a carrier and affect other operating systems which are more vulnerable.

What do you mean by SSH?

SSH stands for "Secure Shell." The SSH protocol is a secure alternative to unsecured remote shell protocols.

What does a Firewall do? How does it work?

A firewall guards your network against attackers. It scans the data that tries to enter the network and imposes restrictions on anything that looks fishy.

Where are the password details kept in UNIX OS?

In UNIX, the details of the passwords are present in the/etc/passwd file.

Linux - File Security

Introduction

Along with windows, iOS and macOS, Linux is also an operating system. An operating system manages the communication between the hardware and software. Linux is open-source, free of cost and highly secure.
In Linux, everything is either a file or a process. Images, texts, videos and hardware drivers, software applications everything is a file. These files are divided into three categories- users, groups and other files which are neither owned by a user nor belong to a group. Permissions are defined for each file i.e. you can define who can read, write or execute the file.

In this article, we will discuss how to secure your files in Linux.

File owner categories

In Linux, a file can belong to any of the following categories -

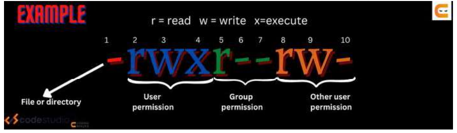
Category	Description	Code
User or owner	A user is the owner of the file. By default, the creator of the file is the owner. A user has read, write and delete permissions.	u
Group	Multiple users together form a group. In a group all the members have equal permissions.	g
Other	A user that is neither an owner nor belongs to a group is kept in this category.	o

File Permissions

Every user, owner or other user has the read, write and execute permission.

Permission	Description	Code
read	Anyone with read permission can view the content of the file but CANNOT edit or modify the contents of the file.	-r or -4
write	Anyone with write permission can edit the contents of the file. At the directory level, anyone with write permission can remove or rename the file.	-w or -2
execute	Only users that have execute permission can run the program.	-x or -1

An example of permissions given to the user, group and other users.
-rwxr--rw-



Let's decode the permissions.

✓ It is made up of 10 characters.
✓ The first character is either '-' or 'd'. For the first place '-' means that it is a file and 'd' denotes a directory.
✓ The characters 2 to 4 are permissions for an owner.
✓ The characters 5 to 7 are permissions for a group.
✓ The characters 8 to 10 are permissions for other users.

So, we can think of permissions as three groups of rwx, where '-' in any place of rwx means the absence of that permission.
In the above example, **-rwxr--rw-** :
1 The permissions are defined for a file. (-rwxr--rw-)
2 The user has all permissions (read, write and execute). (-***rwx***r--rw-)
3 The group has only read permissions. (-rwx***r--***rw-)
4 The other users have read and write permissions. (-rwxr--***rw-***)

As mentioned above, the numeric code for r is 4, w is 2 and x is 1.
The user's rwx permission can be written as (4+2+1) 7.
The group's r-- permission can be written as (4+0+0) 4.
The other's rw- permission can be written as (4+2+0) 6.
Hence, rwxr--rw- can be written as 746.
Use **ls -l** command to view all the ownerships and permissions.

chgrp command

The change group command is used to change the ownership permissions of a group.
→ Syntax:

```
chgrp <new group name> <filename>
```

Note: Only root users have the permission to change the owner or group of the files in the system.

→ Example

```
chgrp newfile ninjafile
```

☑ When we run this command the ninjafile will now become a part of the newfile group.

chown command

This command is used to change the owner of a file.
→ Syntax:
This syntax will change the owner only.

```
chown <new owner> <filename>
```

→ Example

```
chown newowner ninjafile
```

☑ When we run this command, the owner of the file ninjafile will change to newowner.
The following syntax will change the user and the group.

```
chown <new owner name : new group name> <file name>
```

→ Example

```
chown newowner:newfile  ninjafile
```

☑ When we run this command, the owner and the group of the file will change to newowner and newfile respectively.

Changing Permissions: chmod command

If you want to update the permissions of the owner, group or other user, you can use the **chmod** command.
To understand how to use this command, we will use examples –

Adding permissions

There is a file named "ninjafile", with the following permissions: **-rwxrw-r--**
Enter the following command to grant group users permission to execute the file.

```
chmod g+x ninjafile
```

In the above command

✓ 'g' refers to a group.
✓ '+' refers to **granting** permission.
✓ 'x' denotes execute permission.

☑ So, now the ninjafile will have following permissions: **-rwxrwxr—**

Removing Permissions

There is a file named "ninjafile", with the following permissions: **-rw-r--r--**
Run the following command to remove write permission from the owner.

```
chmod u-w ninjafile
```

☑ So, now the ninjafile will have following permissions: **-r--r--r--**
In the above example, '-' sign in u-w is denotes **removal** of write permission from user.

Multiple users permission

There is a file named "ninjafile", with the following permissions: **-rw-r--r--**
Run the following command to **add** write permission to the group and other users.

```
chmod go+w ninjafile
```

☑ So, now the ninjafile will have following permissions: **-rw-rw-rw-**

Separate Permissions to separate users

There is a file named "ninjafile", with the following permissions: **-rwxrw-r--**
Run the following command to **add** write permission to the other users and remove execute permission from the user (owner).

```
chmod u-x, o+w ninjafile
```

☑ So, now the ninjafile will have following permissions: **-rw-rw-rw-**

Permissions to all

There is a file named "ninjafile", with the following permissions: **-r-xr-xr--**
Suppose we want to add write permission to all the categories of users.
Run the following command to add permissions to ALL:

```
chmod a+w ninjafile
or
chmod ugo+w ninjafile
```

☑ So, now the ninjafile will have following permissions: **-rwxrwxrw-**

Multiple permissions

There is a file named "ninjafile", with the following permissions: **-rwxrw-r--**
Run the following command to **add** write and execute permission to the other users and remove execute permission from the user (owner).

```
chmod o+wx ninjafile
```

☑ So, now the ninjafile will have following permissions: **-rwxrw-rwx**

Assigning Permissions

There is a file named "ninjafile", with the following permissions: **-r--rw-r--**
Run the following command to **assign** all permissions to the user and group.

```
chmod ug=rwx
```

☑ So, now the ninjafile will have following permissions: **-rwxrwxr--**

Points to remember about chmod-

✓ Use '+' for adding permissions, '-' for removing permissions and '=' for assigning permissions.
✓ To add/remove/assign permissions to *all* (i.e. to users, group and other) use 'a' (a+rwx).
✓ You can add permissions to multiple worlds by writing them together (like, ugo+x).
✓ You can add multiple permissions to multiple/single world (like ugo+rwx or u-rx, etc).
✓ '=' will overwrite the previous permissions and grant new permissions.

The file Mask

Every new file is given a set of permissions when it is created. A new file is given all the permissions to all the categories i.e. a new file will have -rwxrwxrwx permissions. A new directory is given permission to read and write for all the categories i.e. a new directory will have drw-rw-rw- permissions.
Use **unmask** command to view the permissions given to a newly created file or directory.
Syntax

```
unmask
```

Basic File Management Commands in Linux

File Management Commands

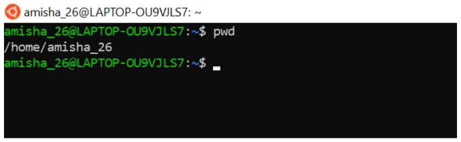
pwd Command

Pwd returns the full path name of the current remote working directory. In the above example, the command output is /home/amisha_26, which implies that we are working in the amisha_26 directory, which is inside the home directory.

Syntax:

$ pwd -L: Prints the symbolic path.
$ pwd -P: Prints the actual path.

Example:



amisha_26@LAPTOP-OU9VJLS7: ~

```
amisha_26@LAPTOP-OU9VJLS7:~$ pwd
/home/amisha_26
amisha_26@LAPTOP-OU9VJLS7:~$ _
```

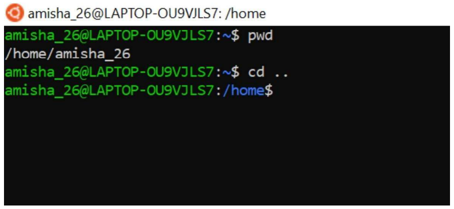
cd Command

The command cd also sometimes called as chdir , it allows you to move between directories. The cd command accepts an argument, generally the name of the folder to which you wish to move. Thus the entire command is cd your-directory.

Syntax :

$ cd <directory_name> (Will move to the desired directory)
$ Cd .. (will move back to the previous directory)

Example:



amisha_26@LAPTOP-OU9VJLS7: /home

```
amisha_26@LAPTOP-OU9VJLS7:~$ pwd
/home/amisha_26
amisha_26@LAPTOP-OU9VJLS7:~$ cd ..
amisha_26@LAPTOP-OU9VJLS7:/home$
```

ls Command

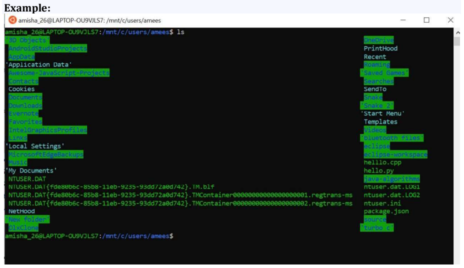
The "ls" command lists files that present the directory you want to work with, except for hidden files.

In the above example, ls list all the files present in the directory named as amees.

Syntax :

$ ls

Example:



amisha_26@LAPTOP-OU9VJLS7: /mnt/c/users/amees

```
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees$ ls
'3D Objects'                                                                                  OneDrive
 AndroidStudioProjects                                                                        PrintHood
 AppData                                                                                      Recent
'Application Data'                                                                            Roaming
 Awesome-JavaScript-Projects                                                                 'Saved Games'
 Contacts                                                                                     Searches
 Cookies                                                                                      SendTo
 Documents                                                                                    Snake
 Downloads                                                                                   'Snake 2'
 Evernote                                                                                    'Start Menu'
 Favorites                                                                                    Templates
 IntelGraphicsProfiles                                                                        Videos
 Links                                                                                       'bluetooth files'
'Local Settings'                                                                              eclipse
 MicrosoftEdgeBackups                                                                         eclipse-workspace
 Music                                                                                        helllo.cpp
'My Documents'                                                                                hello.py
 NTUSER.DAT                                                                                   java-algorithms
 NTUSER.DAT{fde80b6c-85b8-11eb-9235-93dd72a0d742}.TM.blf                                      ntuser.dat.LOG1
 NTUSER.DAT{fde80b6c-85b8-11eb-9235-93dd72a0d742}.TMContainer00000000000000000001.regtrans-ms  ntuser.dat.LOG2
 NTUSER.DAT{fde80b6c-85b8-11eb-9235-93dd72a0d742}.TMContainer00000000000000000002.regtrans-ms  ntuser.ini
 NetHood                                                                                      package.json
'New folder'                                                                                  source
 OlxClone                                                                                    'turbo c'
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees$
```

touch Command

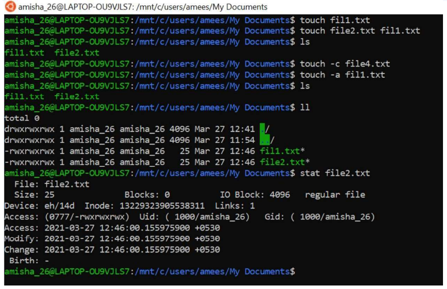
The primary use of the touch command is to alter a timestamp. The function is commonly used for file creation, even though this is not its core role. The terminal program may modify and access times for any specified file. The touch command only creates a file if it does not already exist.

Syntax:

$ touch <options> <file or directory name>

OPTIONS	MEANING
-a	Changes the access time
-c	if the file does not exist, do not create it
-d	update the access and modification times
-m	change the modification time only
-r	use the access and modification times of the file
-t	creates a file using a specified time

Example:



amisha_26@LAPTOP-OU9VJLS7: /mnt/c/users/amees/My Documents

```
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees/My Documents$ touch fil1.txt
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees/My Documents$ touch file2.txt fil1.txt
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees/My Documents$ ls
fil1.txt  file2.txt
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees/My Documents$ touch -c file4.txt
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees/My Documents$ touch -a fil1.txt
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees/My Documents$ ls
fil1.txt  file2.txt
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees/My Documents$ ll
total 0
drwxrwxrwx 1 amisha_26 amisha_26 4096 Mar 27 12:41 ./
drwxrwxrwx 1 amisha_26 amisha_26 4096 Mar 27 11:54 ../
-rwxrwxrwx 1 amisha_26 amisha_26   25 Mar 27 12:46 fil1.txt*
-rwxrwxrwx 1 amisha_26 amisha_26   25 Mar 27 12:46 file2.txt*
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees/My Documents$ stat file2.txt
  File: file2.txt
  Size: 25              Blocks: 0          IO Block: 4096   regular file
Device: eh/14d  Inode: 13229323905538311  Links: 1
Access: (0777/-rwxrwxrwx)  Uid: ( 1000/amisha_26)   Gid: ( 1000/amisha_26)
Access: 2021-03-27 12:46:00.155975900 +0530
Modify: 2021-03-27 12:46:00.155975900 +0530
Change: 2021-03-27 12:46:00.155975900 +0530
 Birth: -
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees/My Documents$
```

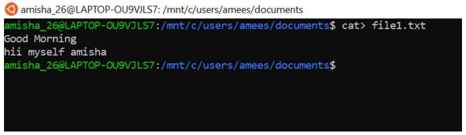
cat Command

Cat or concatenate is often used in Linux. It reads the files' content and gives the content's output on the terminal screen. In the above example, we used the command cat to read the file1.txt and you can see the In the above example, we used the command cat to read the file1.txt and see the output below the command.

Syntax:

$ cat <filename>

Example:



amisha_26@LAPTOP-OU9VJLS7: /mnt/c/users/amees/documents

```
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees/documents$ cat> file1.txt
Good Morning
hii myself amisha
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees/documents$
```

mv Command

The mv command is often used in Linux; this command is used to move a file from one directory to another directory. There is also a second use of the mv command. We can change the name of any existing file to a new name, whichever we desire.

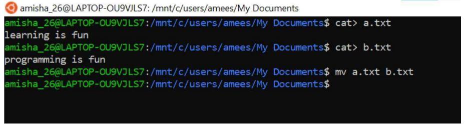
In the above example, we are changing the name of a.txt to b.txt.

Here we are using the mv command, which will change the name of the file a.txt to b.txt.

Syntax:

$ mv <filename> <new file name>

Example:



amisha_26@LAPTOP-OU9VJLS7: /mnt/c/users/amees/My Documents

```
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees/My Documents$ cat> a.txt
learning is fun
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees/My Documents$ cat> b.txt
programming is fun
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees/My Documents$ mv a.txt b.txt
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees/My Documents$
```

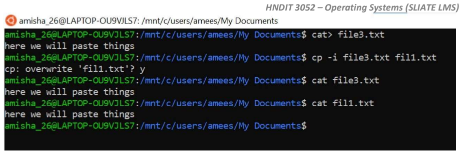
cp Command

Cp means to copy. The command is used to copy files or groups of files. In the above example, we are using the cp command to copy the content of file3.txt and will paste it into the file named file1.txt.

Syntax:

cp <file/directory-sources> <destination>

Example:



amisha_26@LAPTOP-OU9VJLS7: /mnt/c/users/amees/My Documents

```
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees/My Documents$ cat> file3.txt
here we will paste things
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees/My Documents$ cp -i file3.txt fil1.txt
cp: overwrite 'fil1.txt'? y
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees/My Documents$ cat file3.txt
here we will paste things
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees/My Documents$ cat fil1.txt
here we will paste things
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees/My Documents$
```

mkdir Command

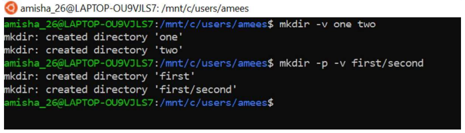
"mkdir" also known as " make directory"**.** This command is used to create single or multiple directories.

Syntax:

mkdir <options> <directories>

OPTION	MEANING
-v	print a message for each created directory
-p	no error if existing

Example:



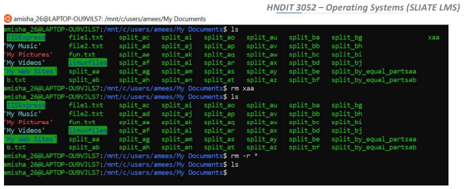
amisha_26@LAPTOP-OU9VJLS7: /mnt/c/users/amees

```
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees$ mkdir -v one two
mkdir: created directory 'one'
mkdir: created directory 'two'
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees$ mkdir -p -v first/second
mkdir: created directory 'first'
mkdir: created directory 'first/second'
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees$
```

rm Command

"rm" also know as "remove"**.**This command is used to remove a file**.** If we use the rm command, the file will be permanently deleted. In GUIs, we got recycle bin and trash for recovering files. So we need to take extra care while using the rm command.

Example:



amisha_26@LAPTOP-OU9VJLS7: /mnt/c/users/amees/My Documents

```
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees/My Documents$ ls
 IISExpress      file1.txt   split_ac   split_ai   split_ao   split_au   split_ba   split_bg                xaa
'My Music'       file2.txt   split_ad   split_aj   split_ap   split_av   split_bb   split_bh
'My Pictures'    fun.txt     split_ae   split_ak   split_aq   split_aw   split_bc   split_bi
'My Videos'      linuxfiles  split_af   split_al   split_ar   split_ax   split_bd   split_bj
'My Web Sites'   split_aa    split_ag   split_am   split_as   split_ay   split_be   split_by_equal_partsaa
 b.txt           split_ab    split_ah   split_an   split_at   split_az   split_bf   split_by_equal_partsab
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees/My Documents$ rm xaa
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees/My Documents$ ls
 IISExpress      file1.txt   split_ac   split_ai   split_ao   split_au   split_ba   split_bg
'My Music'       file2.txt   split_ad   split_aj   split_ap   split_av   split_bb   split_bh
'My Pictures'    fun.txt     split_ae   split_ak   split_aq   split_aw   split_bc   split_bi
'My Videos'      linuxfiles  split_af   split_al   split_ar   split_ax   split_bd   split_bj
'My Web Sites'   split_aa    split_ag   split_am   split_as   split_ay   split_be   split_by_equal_partsaa
 b.txt           split_ab    split_ah   split_an   split_at   split_az   split_bf   split_by_equal_partsab
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees/My Documents$ rm -r *
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees/My Documents$ ls
amisha_26@LAPTOP-OU9VJLS7:/mnt/c/users/amees/My Documents$
```