Operating System Fundamentals

By Dr. Umar Khokhar and Dr. Binh Tran

Georgia Gwinnett College, Lawrenceville (GA)

Chapter 1

Operating System Concepts

Learning Outcomes:

- Definition of Operating System
- Goals vs functions of operating systems
- Computer Systems Organization and storage structure
- Computing Environments

1.1 Definition of Operating Systems

Before we define the term Operating system (OS), we need to define and classify the term software:

1.1.1 Software

The piece of code/program or set of instructions to execute a specific task is called software. The software can be of two types: System Software and Application Software

System Software: are designed to manage the systems and their activities e.g. Operating System and Utility Programs etc.
Application Software: allow the users to interact with the computational systems or system software e.g. Microsoft office suite, Adobe products and video players etc. The application software can be further categorized as web application software (cloud based) and desktop application software.

1.1.2 Operating System

The Operating System (OS) is a system software which:

- Acts as intermediary between the hardware and user.
- Manages the system resources in an unbiased fashion between the hardware and software.
- Provides a platform on which other application programs are installed.

Let us understand, how the operating system acts as intermediary between the hardware and the users. Now the first question is, can we access hardware directly without OS or not? The answer is yes but it would be extremely difficult and inefficient, one has to communicate with computer in machine language to control the complex computational logic. Another example to better understand the role of OS as intermediary is described as follows:
If you want to turn on car Air Conditioner (AC) then can you turn on AC without pressing the AC button? The answer is yes but it would be inconvenient, one has to open the hood of the car and manually turn that machine on. The AC button (on the dashboard of car) acts as interface through which one can access that machine. Similarly, the OS acts as interface between the user and the hardware (CPU, RAM, Hard drive etc.).

OS as Resource Manager: Hardware has many resources e.g. Switches, transistors, CPU, Buses, Memory, Fans etc. If the user wants to directly use these resources (bypassing the OS) then it will be extremely difficult to manage and keep a balance between the devices as well (You need to continuously change the resources requirements e.g. fan speed, memory, CPU utilization etc. depending upon the process).
Example: If you want to watch a movie on your Personal Computer (PC) without having an OS and movie player software using machine language. Then you have to continuously manage the resources e.g. fan rotation, screen resolution, changes of colors, memory requirements and CPU utilization etc. which humanly impossible to manage.

OS as Platform: Since, each application program has its own resources requirement e.g. if we run ten (10) application programs simultaneously then it might create clash of resources for hardware where applications will fight for resources (if the application programs have to direct access to hardware). That is why we install applications on OS which manages the resources and creates a balance between the applications.

1.1.3 Abstract View of Computer System

The computer system is mainly organized into four (4) tiers:

Tier 0: Lowest level, where we have the computer hardware e.g. CPU, RAM, ROM, Hard drive, I/O devices and buses etc.
Tier 1: Operating system
Tier 2: System and Application Programs are located on this tier e.g. Compliers, Assemblers, Device Drivers, Antiviruses and other application programs etc.
Tier 3: Users are located on this tier and if the users wants to get anything done on hardware then they will access it through the application software and OS. For example, if a user wants to watch a movie then the user will go to VLC player (or any other movie player software) and watch the video. The VLC player forwards the instructions (which tell about the resources requirement) to OS and OS will further execute them and allows the access to hardware.

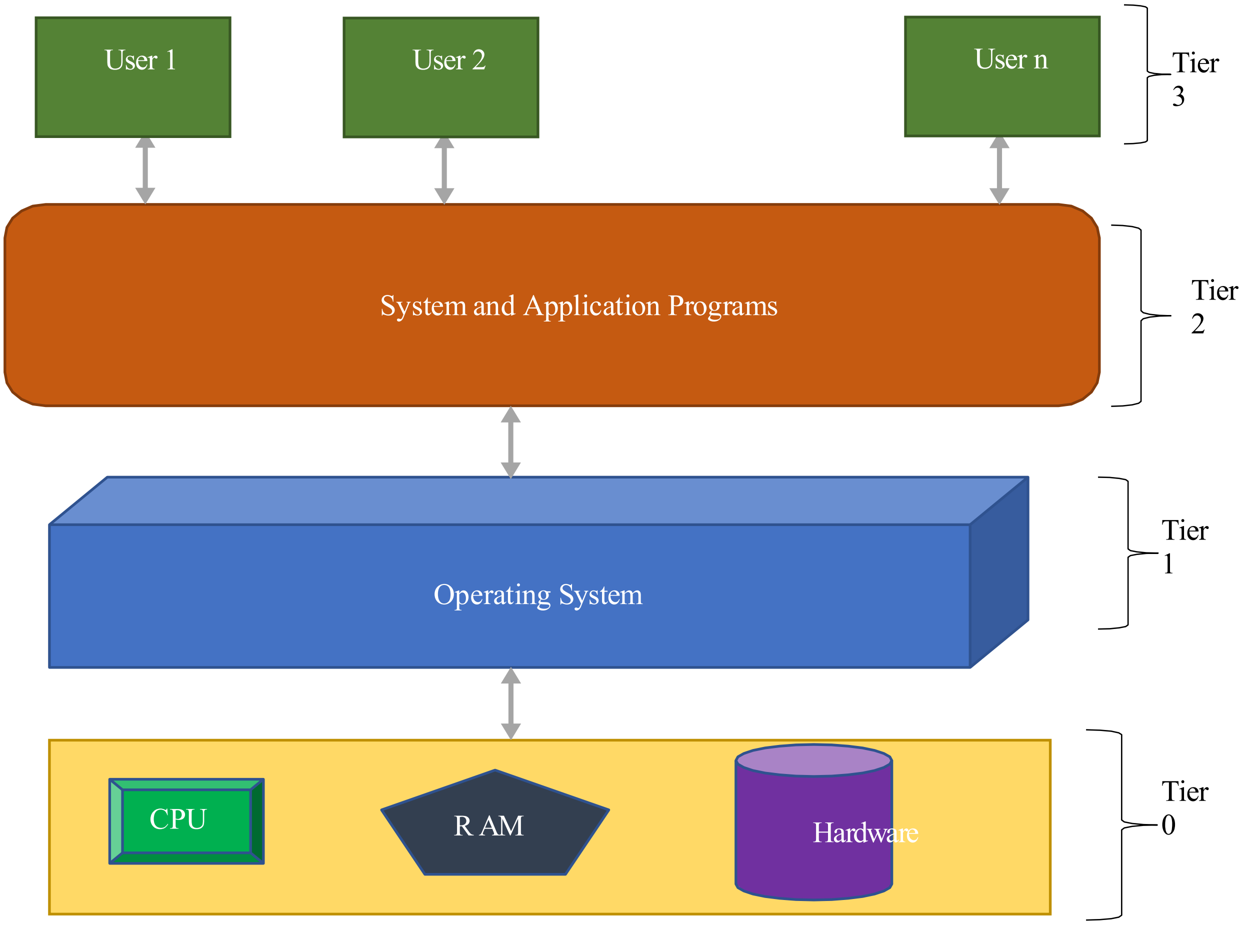


Figure 1.1: Abstract View of the Computer system

1.2 Computer System Organization and Operation

Before we explore the computer system organization and operation, let us discuss some OS related key terminologies.

1.2.1 Kernel

The Kernel is the core component of the OS which helps the users to interact with the hardware. The kernel controls the communication between user and the hardware. If an application software wants to access any hardware or I/O device then it will first request to kernel and kernel will check the resources and allocate the resources or turn down the request.

1.2.2 System Call

If an application software wants to access any hardware e.g. CPU, RAM or ROM etc. then it will request to kernel that request is called system call.

1.2.3 Bit

The smallest entity a computer can process is called bit.

1.2.4 Bytes

Eight (8) bits = 1 Byte

1.2.5 Word

Number of bits a CPU processes simultaneously.

1.2.6 Registers

Smallest storage device which stores the data in bits.

1.2.7 Computer System Components and operation

A modern general-purpose computer consists of following components:

- Central Processing Unit (CPU)
- Device Controller (DC)
- Memory

CPU has two internal parts Control Unit (CU) and Arithmetic Logic Unit (ALU). The CU fetches the instructions from RAM and the ALU firstly executes it and then forward the output to Main Memory (MM) or I/O device.

- The CPU and the DC are connected through common bus which provides the access to shared memory. Each DC is in charge of a specific type of device e.g. disk drivers, audio and video I/O devices etc. The CPU and the DC both can execute in parallel therefore there could be a clash of memory access which can be resolved by memory management controllers. Since, the CPU is much faster than I/O devices therefore to avoid the CPU idle state the DC uses a local buffer which manages the operations in the following fashion:
    a) The DC takes the data/process information from the I/O devices and then stores it in the local buffer.
    b) When the data gets completed then the DC sends an interrupt to CPU so, CPU can dis-

engage itself with the on-going tasks.

c) CPU then moves the data/Instruction from the local buffer to MM and then executes it.

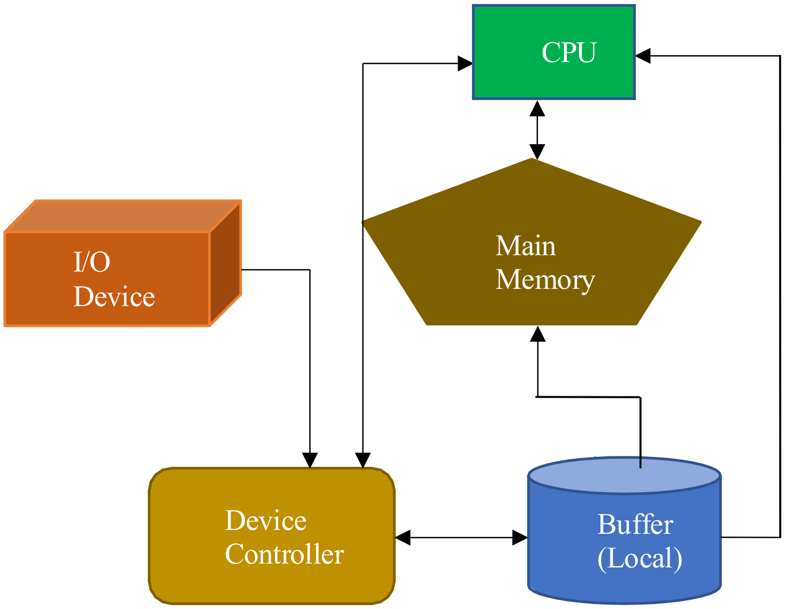


Figure 1.2: Computer System Operation

1.3 Goals vs functions of Operating Systems

The primary goal of general-purpose OS is convenience (user friendly) while the efficiency is considered as a secondary goal. However, it is other way around for Network OS or any other server-based computing environment.
There are six (6) main functions of a general-purpose OS:

- Process Management
- Memory Management
- I/O Device Management
- File System Management
- Network Management
- Security and Protection

The detailed description of these functions of OS is presented as follows:

1.3.1 Process Management

A program/software does nothing unless its instructions are executed by a CPU so, a program in execution is called process. A process needs certain resources to accomplish its tasks e.g. CPU, RAM, I/O devices and these resources are allocated to the process while it is being executed. The OS reclaimed all resources when the process gets terminated or completed. So, the program itself is not a process but a passive entity whereas the process is an active entity (you cannot carry a process on storage media but can store a program). A process can perform multiple tasks simultaneously using either the parent-child model or threading. The detailed description of both of these models are discussed in Chapter 3. The single threaded processes have one Program Counter (PC) which tells the complier about the next instruction to be executed, however the multithreaded processes have one PC/thread. The OS does the following to manage a process:

- Execute
- Create and Terminate
- Abort

1.3.2 Memory Management

To execute a program all of the instructions and the data must be loaded onto RAM which further provides instructions to CPU to execute a process. The Memory management activities are:

- Allocating and de-allocating Memory
- Assigning priority to processes
- Keeping a log/track of memory utilization

1.3.3 Storage Management

Logical storage unit is called "*File*" which is mapped onto the physical media. To store a data onto the physical media, the data must be in the form of a file. For managing storage:

- File System Management
- Mass Storage Management
- Caching

1.3.3.1 File System Management

The most visible component of an OS is the file which is the collection of related information defined by its creators and organized in directories.
If the multiple users have access to the files then it is important to define access controls of the file. The OS manages the file systems using filing methods e.g. FAT, NTFS and XFS etc. These filing methods defines the file creation, deletion, location and back up methods of various file formats.

1.3.3.2 Mass Storage Management

As we know, the RAM (Random Access Memory) is too small to accommodate all of the data and the programs and moreover volatile, therefore, we use secondary storage to back up the main memory. Besides the user's data, compilers, assemblers, word processors and formatters are stored on disk until located onto the RAM. The mass storage mainly manages the free space and storage allocation.

1.3.3.3 Caching

The cache is the small memory which mainly assists the operations of the RAM and unlike RAM it is directly located onto the CPU. The cache holds the temporary data/instructions on it and saves the time to fetch the instructions over and over from the RAM. The detailed description and the working of the cache is discussed in chapter 7.

1.3.3.4 I/O subsystem

The I/O subsystem manages the memory and the device drivers optimally. The I/O subsystem also manages the resources to facilitate the I/O subsystems.

1.3.4 Protection and Security

The protection is the mechanism for controlling access of processes/users to the resources (e.g. which process can access which resources and how much it can access etc.). The security module acts as defense of the system against internal and external attacks.
The Protection and the security module can distinguish among the users based on the user ID, passwords and can define the access rights for each user of the systems.

1.4 Comparison of the Computing Environments

Different devices have different computational requirements and the power constraints, which is why we need different operating systems for various types of computing environments. The summary of different computing environments is presented in Table 1.1.

Computing Environments	Entities/Components
Traditional Environment	1. Personal Computers 2. Desktop & Laptop **Examples**: Windows, MAC or Linux
Mobile Environment	1. Handheld Devices such as smart phones and tablets **Examples**: IOS & Android etc.
Network Environment	1. Distributed • Collection of Heterogenous networks 2. Client-Server 3. Peer to Peer **Examples**: Microsoft Server, Linux & Unix etc.
Virtualization	1. Allows guest Oss on same/different host OS. 2. Allows shared CPU & Memory 3. Requires virtual machine manager (creates a bridge b/w Oss). **Examples of Virtual Software**: Virtual Box, Vmware and Dockers
Real Time Embedded Systems	1. System of Chip (SoC) 2. Computer which is the part of another computer. **Examples**: Windows 10 IoTs, RTOS etc.
Cloud Computing	1. Cloud: Network of servers hosted over internet and use of cloud for storage, processing and executing data/app is called cloud computing. **Examples**: IOS & Android etc.

Table 1.1: Different Computing Environments

Chapter 2

Operating System Structures

Learning Outcomes:

- Operating System Services
- System calls and its types
- System Programs
- Various OS Structures

2.1 Operating System Services

OS provides many services for the execution of the programs. Although, these services vary from OS to OS but there are few mandatory/common services which an OS should offer. The detailed description of these services is presented as follows:

2.1.1 User Interface

The User Interface (UI) is a place through which, user can give input/interact with the systems. There are mainly three UI methods, a typical OS offers:

- *Command Line Interface (CLI)*: Uses commands (entered through keyboard) to interact with the applications and computer programs. In some OSs (some of the Linux flavors) the command interpreter is included in kernel while in the windows & Unix, it is treated as special program. The main function of the CLI is to get & execute user-specified commands e.g. create, delete, list, print and copy etc. Usually, the command interpreter doesn't understand the command in a way, it just uses the command to identify the file (system) to load into the memory and execute it. e.g. ***del file.txt,*** It will search file ***del*** into memory & execute it with the parameter file (file.txt).
- *Batch Interface:* Commands are typed in files (using scripting languages) and then these files are executed to interact with applications and programs e.g. Batch scripting (windows) and Bash scripts (Linux) etc.
- *Graphical User Interface (GUI):* User can interact with the applications using pointing devices or keyboard. GUI offers the user-friendly environment, where user can launch programs with the input devices.
  Many OSs, provide both the CLI and GUI e.g. Microsoft (windows) offers GUI + Command Line, MAC offers GUI + Terminal (bash) and Unix & Linux offer GUI + Terminal.

2.1.2 Program Execution

OS must be able to load program instructions into the memory to run the program activities.

2.1.3 I/O operations

Operating system should provide mechanism to control I/O devices.

2.1.4 File Systems

Programs need to read, write files and manage directories, moreover file access control is also required therefore file system (e.g. FAT, NTFS and XFS etc.) is also one of the important OS services.

2.1.5 Communications

Communication protocols and modules are also required which let the processes to communicate with other processes or computers with other computers on the network.

2.1.6 Error Detection

Many errors can occur in CPU & memory or even in I/O devices (Power Failure, Memory error, lack of papers in printers etc.). Therefore, an error detection and correction module are an integral service of an OS.

2.2 User Mode vs Kernel Mode

When we run any application, we are in the user mode but as you know all the resources can be controlled or accessed through Kernel. So, when a user program needs to access hardware (CPU, RAM, Hard Disk or I/O devices etc.), it makes a system call and go to the Kernel mode.

Example: *When we open an Excel sheet, we are in the user mode, once you update some data on the sheet and want to save this change on Hard disk, then the application (MS-Excel) will make a system call to kernel, then we will enter in Kernel mode and the result will be stored on the hard disk. After getting the task, you will return to user mode. Similarly, if you want to print a number or data on monitor (e.g. In Java scripting, system.out ())then this request will go to Kernel and through kernel, the user will access the monitor and the result will be displayed on monitor.*

2.3 System Call

To understand system call, let us take an example:
When we run a program then its instructions get loaded into the RAM and it becomes a process. Now, if the process wants to access any I/O device then it will request to Kernel, this request is call system call. These system calls are designed in High Level Language (e.g. C and C++ etc.).

2.3.1 Types of System calls

Since, a user needs to access many resources, the types of system calls depend on the use of these resources. There are six main categories of system calls:

a. Process Control
b. File Manipulation
c. Device Manipulation
d. Information Maintenance
e. Communication
f. Protection

In a general-purpose OS, there usually more than different hundred (500) system calls e.g. Windows-10 has more than two thousand (200) different system calls.

a) **Process Control**: The Process control system calls mainly manages the processes e.g. process creation, termination, loading, aborted and deleted etc. some of the process related system calls are:
fork( ): Create a child process
exit( ): Terminate a process
Kill ( ): Terminate a process abnormally
Nice ( ): Increase a priority of a process

b) **File Management**: The file management system calls are used to create, delete, open, close, read and write the file. Some of the file management system calls:
create( ): To create a file
open( ): To open the file
Write ( ): Write filr
mkdir ( ): Create a new directory

c) **I/O devices Management**: The I/O devices management system calls are used to control the devices and allows the users to read and write the data from devices.

d) **Information**: If we need to get information about the devices or processes then we use information related system calls e.g.
Getpid( ): To find the id of the process (metadata of the file size, permission and types etc.)
Set timer( ): Set time and data on the computer

e) **Communication:** The communication related system calls are used to create and delete the communication and forward/receive messages.

f) **Protection:** Provides the mechanism for controlling access to the resources. Some of the protection related system calls are as follows:

Set_permission ( )
Get_permission ( )
Allow_ user ( )
Deny_user ( )

2.4 System Programs

So far, we have discussed that we need system calls for the implementation and the working of the OS (services of the OS). The system programs are basically utility programs which help/provide the interface to user to create a system call. The system programs are called system utilities.

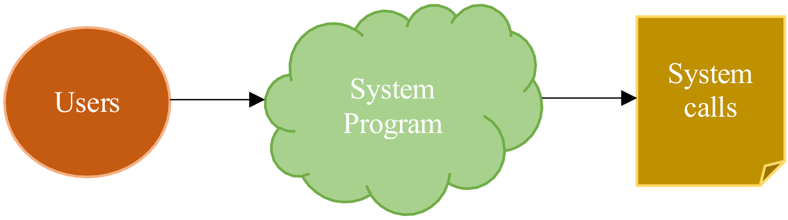


Figure 2.1: System Programs Overview

System programs are divided into six categories. The detailed description of these system programs categories is presented as follows:

a) **File Management:** Create, delete, copy, rename and prints.
b) **Status Information:** Sometimes, we need to find information about the file size, time (when the file was created, Hard drive space etc.) then we use status information program.
c) **File Modification:** Use text editors to:
    1. Create and modify the files
    2. File searching (Bar to search the file)
d) **High-level language support**: OS provides small system programs such as compilers, assemblers, debuggers and interpreters which converts high-level programming languages into low-level programming so, hardware can understand the instructions.
e) **Program Execution:** OS also offers system programs like linkers and loaders which loads the program instructions onto RAM and then forwards to CPU.
f) **Communication:** The communication system programs enable the TCP/IP protocol and allows the sockets to communication with other computers over the network and internet.

2.5 OS Design and Implementation

To design and implement an OS, there are mainly three (3) steps:

- Design Goals
- Mechanism and Policies
- Implementation

Design Goals: First problem in designing an OS is to finalize the design goals and specifications. There are two types of design goals:

a) User Goals: Goals of the users e.g. user convenience, secure & fast and easy to learn

b) System Goals: Goals of the system developers e.g. reliable, flexible, easy to design and upgradable etc.

Polices: What to do (What OS will be doing)?
Mechanism: How to get it done (Development of Algorithms and Mathematical Modeling)?

Implementation: In the earlier OS, the assembly and system programming language were most commonly used programming language to design an OS. Then high-level programming language "Basic" have been used for long for OS designing and implementation. Now a days, we use a mixture of the programming languages for implementation of OS e.g.

- Lowest level (CPU and other hardware components/ interface) ➔Assembly
- Main Body ➔ High level Programming languages e.g. C/C++/Basic/Java
- System calls ➔ High level Programming languages

The high-level programming languages are easier to configure but do slower execution.

2.6 OS Structures

The general-purpose OS is a very large program e.g. Windows 11 have roughly 50 Million lines of code. A proper structure of the OS would be extremely helpful in designing and organizing the various functions of the OS. We have discussed the five (5) OS structures:

1) Simple
2) Layered
3) Microkernel
4) Modules
5) Hybrids

2.6.1 Simple

In earlier computers, there was just one CPU, small RAM and few I/O devices. Therefore, a simple structure also known as Monolithic can be used to design an OS which can optimally manage the OS functionalities. In Monolithic structure, all the components of OS e.g. Process management, Memory Management, device management and File management etc. are housed under one single unit.
There is no concept of the modules so, if the CPU wants to fetch any data (for any process of program) then it will be referred to the main program of the OS to pick up the data. The figure 2.2 presents the monolithic OS structure.
One of the biggest advantages of the monolithic structure over other structures is fast execution of the requests. However, since the user's programs and the OS files are co-located therefore it is more prone to errors and viruses, since the user's programs can bring errors inside the Kernel space. Moreover, the upgradation of the OS is also a cumbersome process since, the whole body/code will need to be replaced/updated. One of the examples of the simple OS structure is MS-DOS.

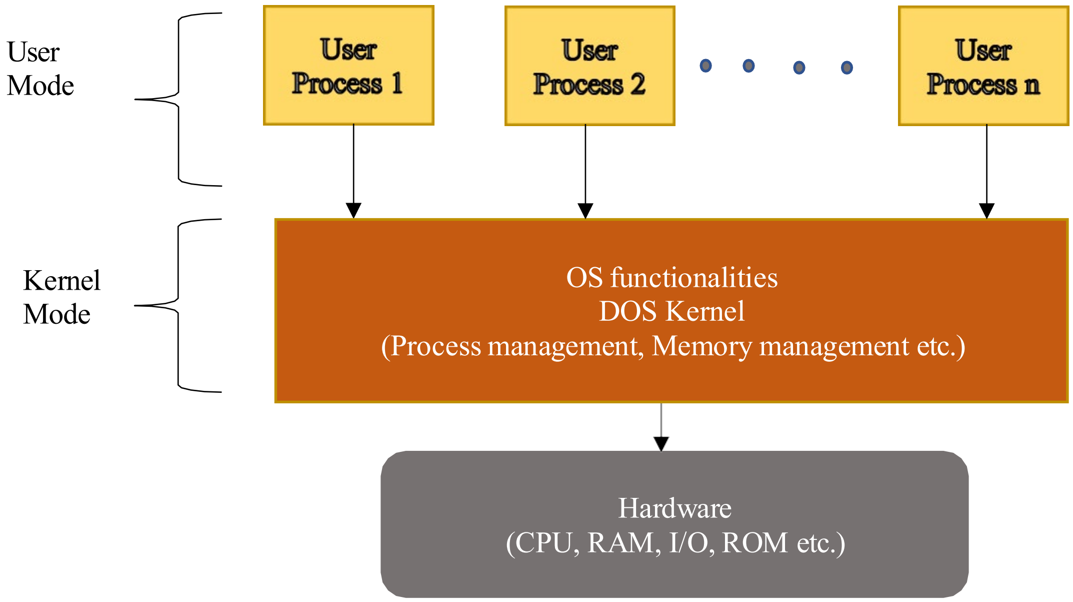


Figure 2.2: Monolithic Structure of OS (MS-DOS)

2.6.2 Layered Structure

In layered structure, the OS consists of several layers, where each layer has aa well-defined functionality, which is coded and tested independently. The lowest layer (layer 0) interacts with the underlying hardware and the topmost layer (layer N) provides an interface to the application programs and the user process. The figure 2.3 presents the layered structure of the OS.
Each layer relies on the service of the layer above it and the communication takes place between adjacent layers only. MULTICS is one of the OSs which entirely uses the layered structure in its design.
There are two advantages of layered structure over the monolithic are:

- Easiest way to add new features (add/replace layer without effecting the other layers).
- Easy to create new layers.
- OS update is also convenient since, only the layers (code) will be replaced not the whole OS program.

2.6.3 Microkernel Structure

The main idea is to keep minimum functionality within the kernel but providing all the OS functionalities through Kernel. In other words, the critical functionalities of the OS e.g. Process Scheduling, Inter-process Communication (IPC), Memory Management, Thread Management and Interrupt Management etc. are kept in kernel while other OS functionalities e.g. File management, I/O management and networking etc. are not included in kernel. These functionalities are implemented through separate server process. These server processes are placed in user address space. When the user application needs to access any server process then it will make system call to IPC which will forward the request to the concerned server process.
Just like layered structure, the microkernel also offers easy upgrade and proves to be reliable (if any of the server process fails then it can be separately updated. The figure 2.4 presents the microkernel structure of the OS.

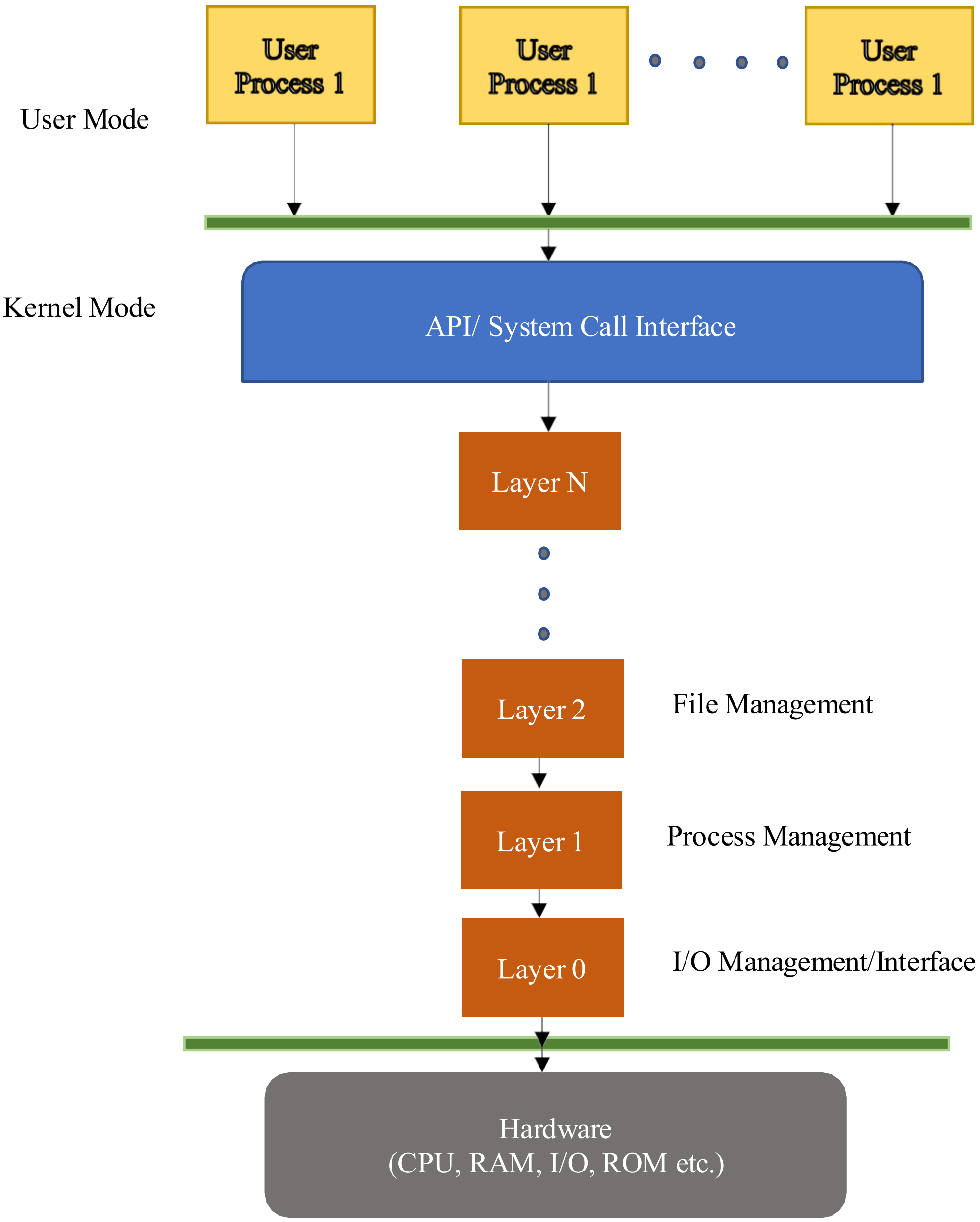


Figure 2.3: Layered Structure of OS (MULTICS)

2.6.4 Modules

Just like layers, the OS can be designed using loadable kernel modules. These modules are independently coded but offers more flexibility than layers.
Unlike the layers, the modules can communicate to other modules as well. The Linux and Solaris use the modules structure of the OS.

2.6.5 Hybrids

Most of the Modern OSs use the hybrid structures (use more than two models) e.g.

- Linux & Solaris [Monolithic + Modules]
- Windows [Monolithic + Microkernel]
- MAC [Microkernel + Layered]

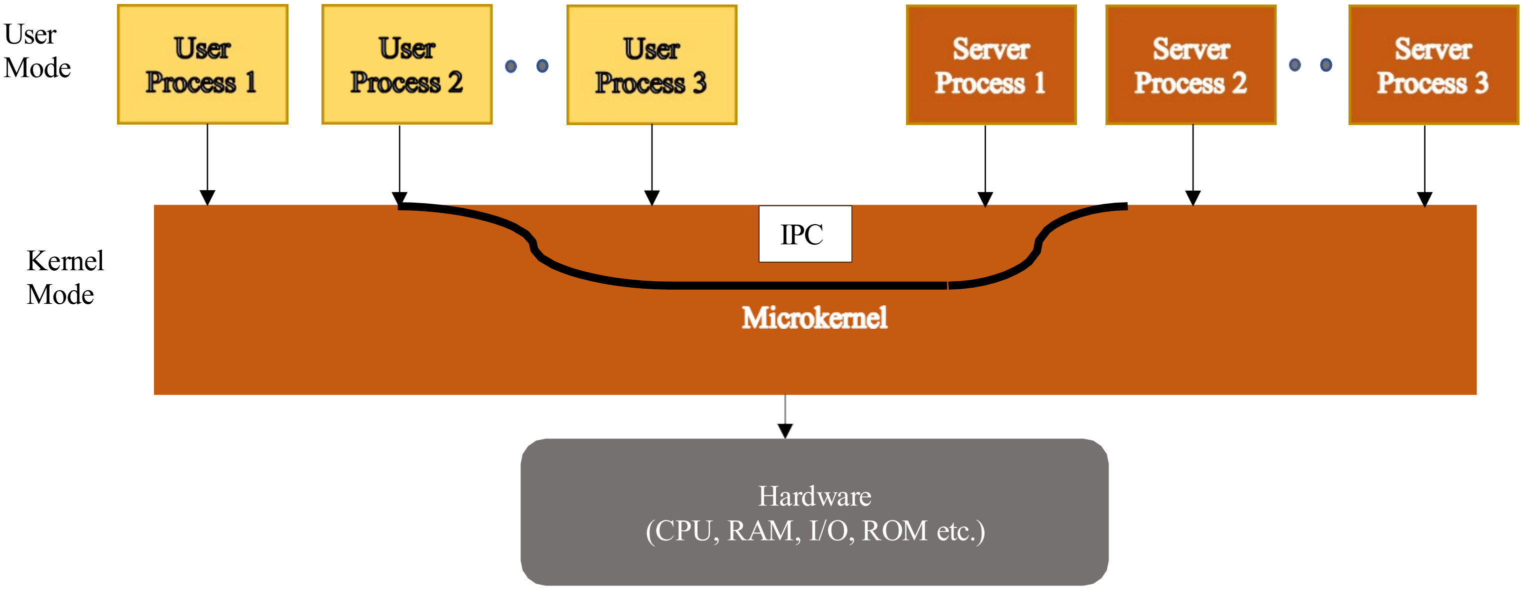


Figure 2.4: Microkernel Structure of OS

2.7 Modern OS Structures

In this section, we will be discussing structures of three (3) modern OSs; MAC, IOS and Android. The detailed description is presented as follows.

2.7.1 MAC OS

Just like any other OS, the MAC OS is divided into two parts; User address space and Kernel. In the User address space, the top most layer offers the framework of GUI (AQUA) and then MAC uses Cocoa API (an Object-oriented framework) for interaction wit kernel. The Kernel environment involves the BSD (Berkley Software Distribution) and the MACH. The MACH provides the memory management and IPC etc. while the CLI (Command Line Interface) & Networking routines uses BSD framework. The figure 2.5 presents the MAC OS structure.

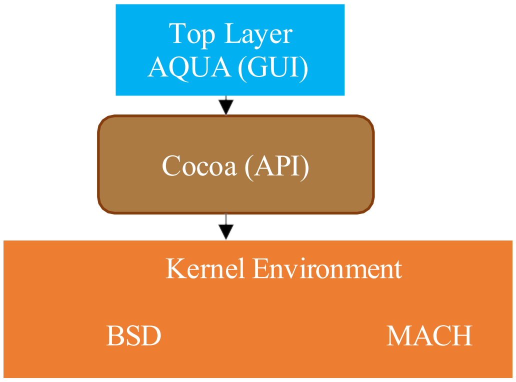


Figure 2.5: MAC OS Structure

2.7.2 IOS

The IOS (Apple Inc.) involves four layers; Cocoa Touch, Media, Core services and Core OS. The first two layers reside within the User Address space and the other two layers are located onto kernel. The Cocoa touch provides API and also supports the touch screen interaction. The media layer provides the graphics and manages the audio and video device controllers. The core services layer provides the device connectivity to the iCloud (cloud computing) and the core OS layer offers other OS functionalities e.g. Process management, Memory management and network routines etc. The figure 2.6 presents the IOS structure.

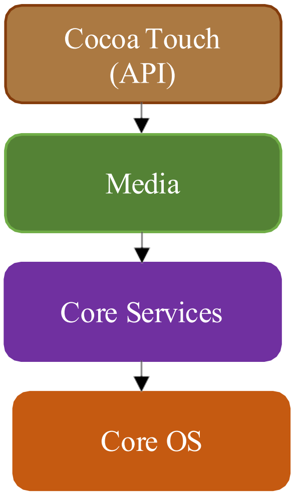


Figure 2.6: Conceptual view of IOS Structure

2.7.3 Android

The android structure is quite similar to IOS (layered stack) where at the bottom of the stack, it uses the Linux Kernel (customized by Google), which manages the process, memory and devices etc. Google has made a special API for application development where the Java files are first compiled to byte code and then executable.

2.8 Operating System Debugging

The debugging module is one of the most important components of the OS which finds and corrects the errors in the hardware & software. There are two components of the debugging module; Failure Analysis and performance tuning.

a) Failure Analysis

If a process fails, an OS writes the error information to a log file to alert the users that error has occurred.

- *Core Dump*: OS captures the memory of the process and store it in a file for the later analysis.
- *Crash Dump*: If the kernel fails it is called crash and the error log file is called crash dump.

The firmware updates use these logs and fixes the errors.

b) Performance Tuning

The users can see the performance of the various hardware resources and also manages the various tasks and processes using the performance tuning utilities e.g. windows task manager (MS-Windows) and activity monitor (MAC).

2.8.1 Emulator

The part of the OS which makes one component of OS to behave like another component.

2.8.2 Operating System Generation

Usually, OSs are designed to run on any of the class of the machines. The SYSGEN module in OS obtains information about the hardware configurations of the device. In other words, the SYSGEN tests the hardware configurations and see whether the OS can be installed on the device or not.

Chapter 3

Processes

Learning Outcomes:

- Definition and scheduling of the Processes
- Operation of the processes
- IPC methods and mechanisms

3.1 Definition of the Process

A program in execution is called process. When we run a program then its instructions get loaded onto the RAM and that instance of the program is called the process. The process is divided into four components/sections which is presented as follows:

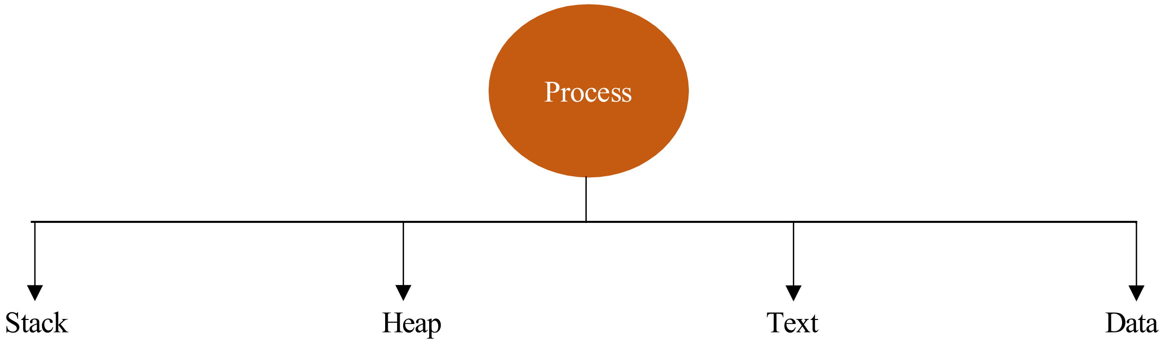


Figure 3.1: Components of the Process

- **Stack:** Holds the temporary data e.gg. functions, parameters and local variables.
- **Heap:** will dynamically allocate the memory to the process during the run time.
- **Data:** contains global variables and static variables.
- **Text:** includes program counter and the contents of the process registers.

3.1.1 Process State

When a process executes then it changes its states (during different tasks, it task get change then the state of the process will be changed). The process goes through the following five (5) states:

- **New:** When process is created.
- **Running:** Instructions of the process are being executed.
- **Waiting:** Waiting for some event to be occurred (Response from Hardware).
- **Ready:** Event has occurred then process is waiting in ready queue to be assigned to process.
- **Terminated**: Process execution has completed.

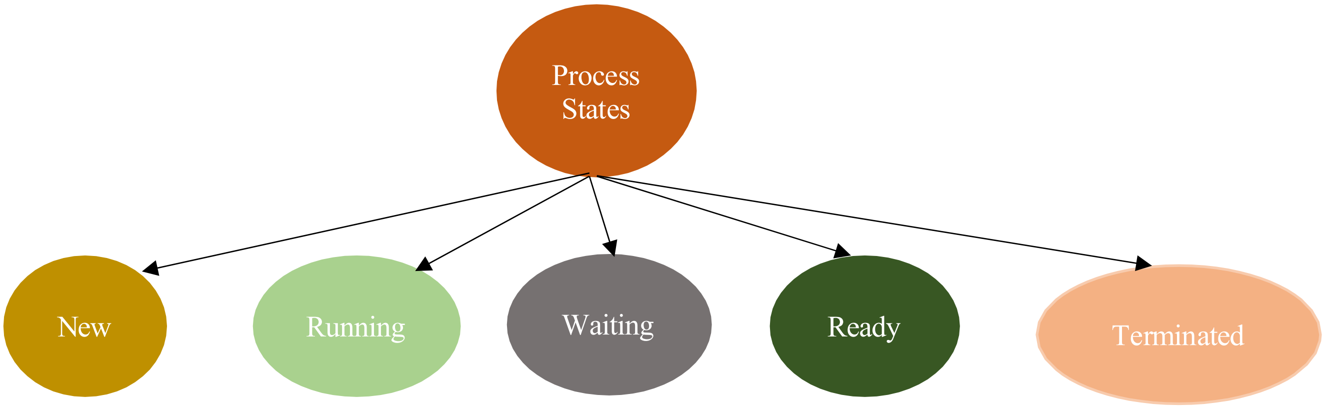


Figure 3.2: Components of the Process

3.1.2 State Diagram

CPU executes only one process at a specific instant of the time (running state), however multiple processes can stay in ready and waiting state. Let us understand, how a process goes through the different states during its lifetime:

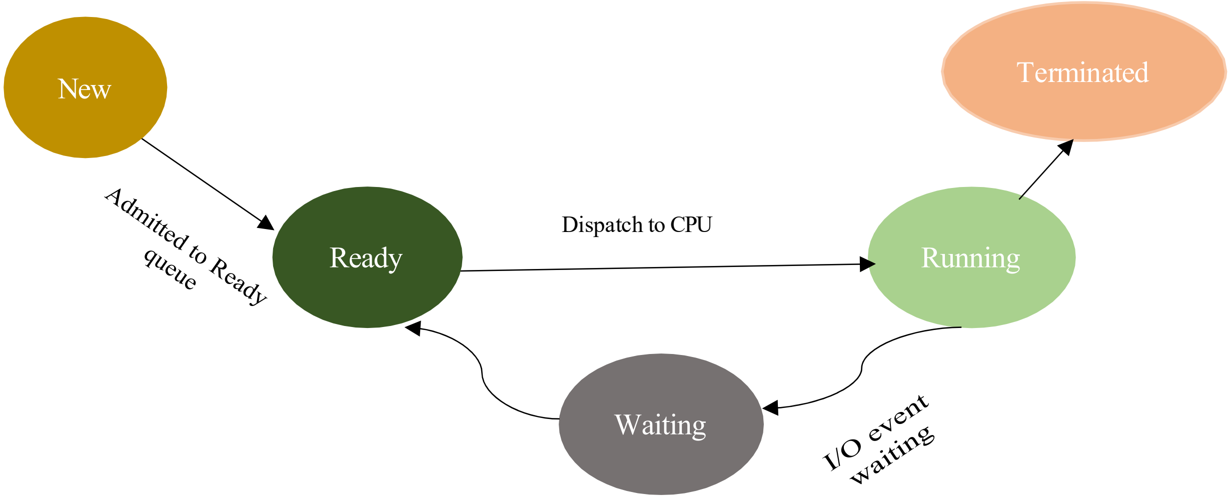


Figure 3.3: State Diagram of Process

1) When we create a process, it is in “*new*” state.
2) Then it goes into a “*ready*” state, where the process waits for its turn for execution (Processes follow FIFO (First In First Out) algorithm and joins the ready queue at the end of the queue).
3) When the process turn comes then a dispatch operation moves the process from “*ready*” to “*running*” state.
4) Once the process gets completed (finished executing its computations) then it goes to “*termination*” state.
5) If a process needs to access I/O device then it goes to the “*waiting*” state and remains in that state unless I/O event occurs. After the I/O event occurs then the process rejoins the “*ready*” state and on it turns moves to “*ready*” state.

3.1.3 Process Control Block (PCB)

Each process has a PCB and the PCB is the data structure maintained by OS for every process. The PCB holds the detailed information about the process. The PCB includes the following details of the process:

ID: The identity of the process, so the process can be distinguished (e.g. P1, P2 etc.)
State: In which state the process is currently located (e.g. New, Running etc.)
Pointer: The connections of the process (for which we are designing this PCB) e.g. it is connected with which child or parent process.
Priority: Let say, we have five processes (P1, P2 …P5) then what will be the priority of the process.
Program Counter: Points to the next instruction.
CPU Register: Holds the information about the registers which will be used by the process.
I/O Information: Devices required/used by the process.

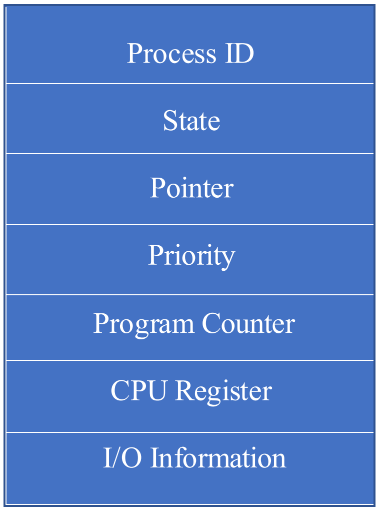


Figure 3.4: Data Structure of PCB

3.1.4 Thread

Thread is a lightweight process which is the basic unit of the CPU execution. The process can have many threads which execute together and offer multiprocessing environment. Each thread has its own ID, stack and the set of the registers. The threads are mainly of two types; User Level Thread (ULT) and Kernel Level Thread (KLT). We will discuss the threads in detail in chapter 4. The ULTs cannot make system calls and if they need to access any resource then it requests through the process.
Each thread has its own code, data and set of registers which they don’t share with other registers.

3.2 Process Scheduling

CPU executes one process at a time and it allocates same time slice to each process. If any process couldn’t get completed during that time slice then the process moves to blocked queue (waiting state). The figure 3.5 describes the working of the process scheduling:

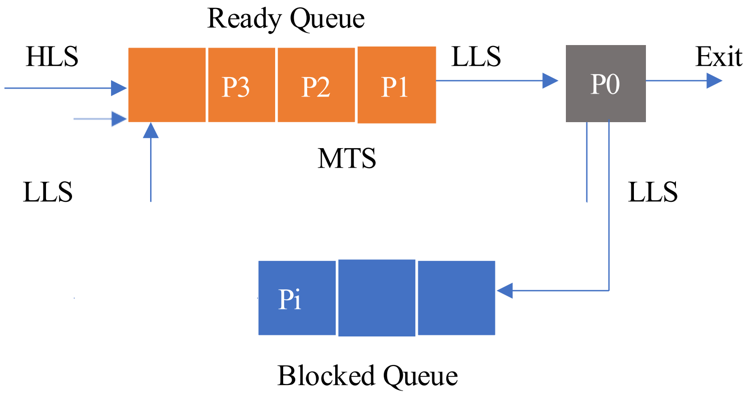


Figure 3.5: Process Scheduling

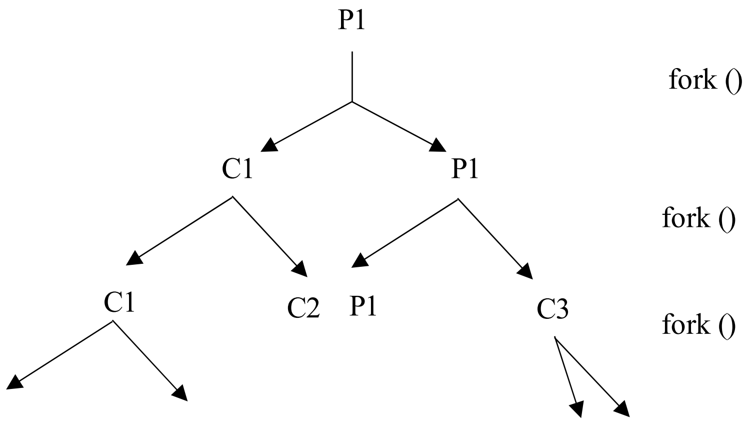
The process scheduling steps are presented as follows:

1) The High-Level Scheduler (HLS) is responsible for admitting new processes into the ready queue. The HLS is also termed as Admission Scheduler or Long-Term Scheduler.
2) The Low-Level Scheduler (LLS) moves the processes from ready queue (if CPU becomes available) to running state (CPU). It allows the other processes to advance (moves forward) in the ready queue, so, mew processes will be admitted to the system. The processes can be of two types; CPU bound and I/O bound. The CPU bound processes (once enter in the running state) execute and then get terminated. The I/O bound processes (once enter in the running state) moves to blocked queue (waiting state) and joins the blocked queue at the end. The process waits for I/O event to be occurred and once the I/O event occurs then LLS moves the process back to ready queue and then LLS moves it to running state.
3) Each process gets same time slice and stays in the running state for 10-100ms. If the process didn't get complete during that time slice then its state will be stored in the memory and LLS will move it out and put it back to the ready queue. When the incomplete process turn comes then the CPU restores its original state. The restoration of the previous state of the process is called context switching (context switching takes 10 microseconds).

3.3 Operations on Processes

There are two main operations on processes; Process creation and Process termination

1) **Process Creation**: When we run a program (execute) then it becomes a process (process is created). This main process is called parent process, which can create child processes. Each child has its own process ID and usually parent and children can share resources and can execute simultaneously. When the parent creates child then that child creates more children and eventually form a tree of processes. To create a child, parent use fork ( ) system call and the child will be the clone of the parent but have its own process ID and PCB. The detailed description of the process creation is presented as follows:



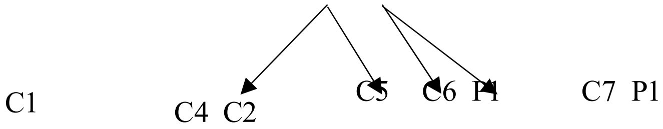

C1
C4
C2
C5
C6
P1
C7
P1

Figure 3.6: Process Creation Tree

$$Number\ of\ Processes = 2/$$

$$Number\ of\ child\ processes = 2/ - 1$$

Where n is the number of fork ( ) system calls.

2) **Process termination:** A process terminates when it has finished executing its last statement. The processes use exit ( ) system call to tells the OS to terminate itself. Moreover the parent process can also terminate child processes using abort ( ) system call. There are two main reasons when parent wants to kill the child:
   - Child has exceeded allocated resources.
   - Task assigned to the child no longer required.

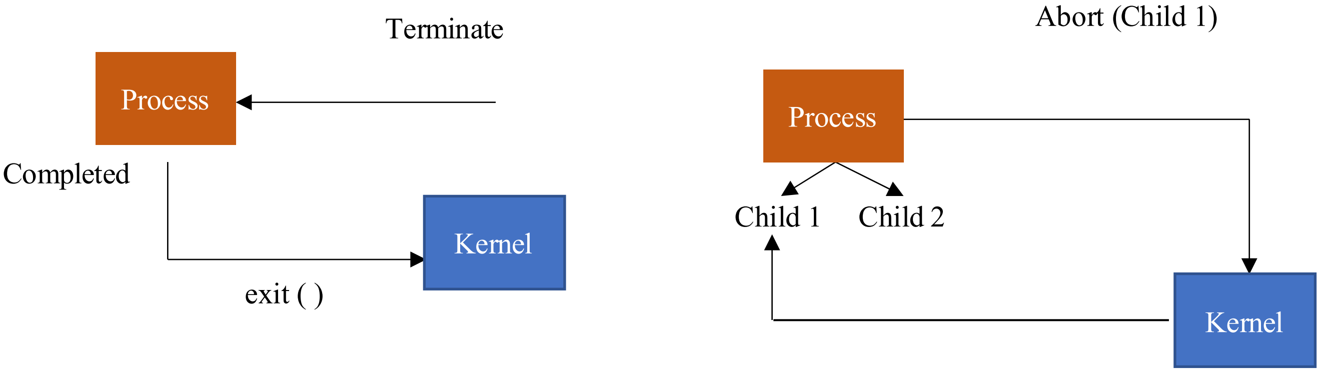


Figure 3.6: Process Termination

3.4 Inter Processes Communication (IPC)

IPC is a mechanism which allows one process to communicate with other processes and also provides synchronization. The processes can be; Independent or Cooperating processes.

- *Independent Process*: The process which doesn't affect other processes or cannot be affected by other processes.
- *Cooperating Process*: Process which can get effected by other processes or can be affected by other processes are called cooperating process.

The cooperating processes allow the information sharing, improve the computational speedup, modularity and allows the user to execute multiple processes simultaneously. The cooperating processes use two IPC models; Shared memory model and Message passing model. The figure 3.7 presents the IPC model in detail.

3.4.1 Shared Memory Model

A specific region of the memory is established which then two or more processes start sharing. Assume that process (P1) has used some resources or performed some functions /fetched information then the process (P1) will make a log in shared memory. Now, if the process (P2) wants to fetch information or access resources then firstly it will check the shared memory before requesting to the Kernel which makes shared memory faster than message passing model. The shared memory resides in the address space of the process who first initiates the communication e.g. if the process (P1) initiates the process first then memory will be located in the Process (P1) address space.

To better understand the shared memory model, let us consider the producer-consumer model.

Producer: produces the information which will be consumed by the consumer.

Consumer: consumes the information which will be produced by the producer.

The producer can be of two types:

a) **Unbounded buffer**: buffer of the memory is unlimited; the producer can produce the information without the consumer consumption.

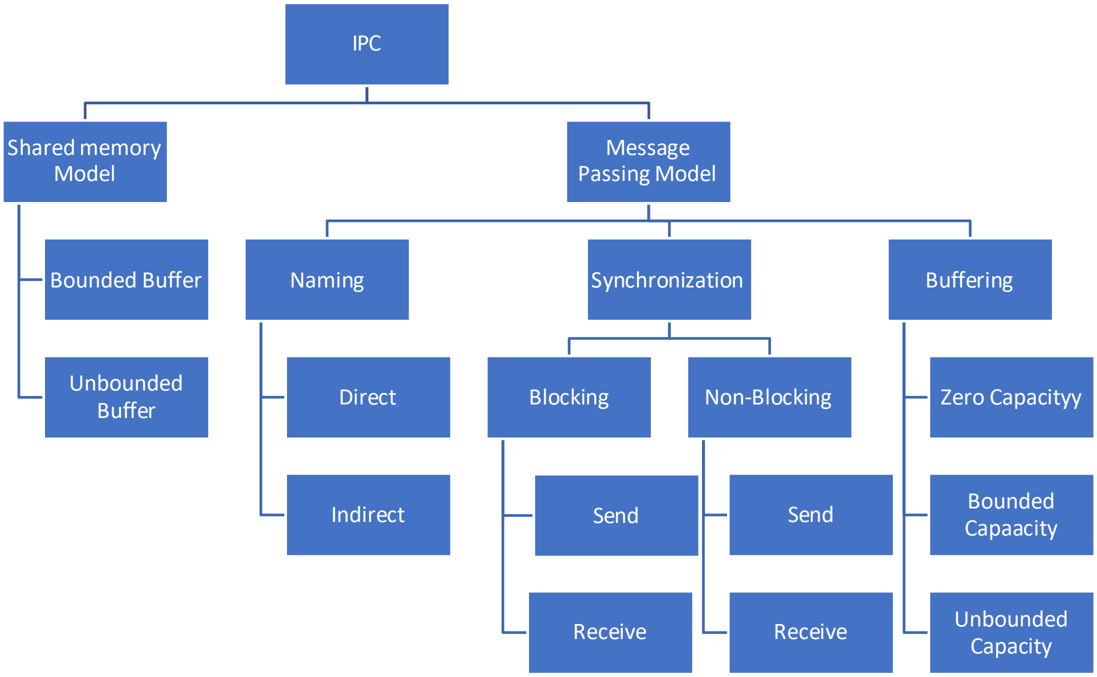


Figure 3.7: IPC Models

b) **Bounded Buffer**: as name suggests, the producer contains limited number of buffer and if the buffer gets full then producer will wait (stop producing the information) unless the consumer consumed it. (e.g. if we have three (3) buffers and all of them get full then producer will not produce).

3.4.2 Message Passing Model

The cooperating processes communicate by exchanging messages with each other and easier to implement. The message passing model is more useful in distributed environment (LAN, MAN and WAN). There are three (3) message passing models; Naming, Synchronization and buffering.

a) **Naming**: Both the sender and the receiver should know each other's identification. The naming model can be implemented using direct and indirect communication.

i. *Direct communication*: Both the sender and the receiver have direct communication link and can send and receive messages directly from each other.

$$Send\ (P, msg) = Send\ message\ to\ P$$
$$Receive\ (Q, msg) = Receive\ message\ from\ Q$$

Where P and Q are the IDs of the processes.

ii. *Indirect Communication:* Firstly, the OS creates a Mailbox (M) then the process send/receive messages and finally delete the Mailbox (M).

$$Send(M, msg) \rightarrow Send\ to\ MailBox$$
$$Receive\ (M, msg) \rightarrow Receive\ from\ MailBox$$

In the indirect communication model, there might be a situation where two or more processes might try to access the same Mailbox and can create a deadlock situation. This deadlock can be avoided by using multiple Mailboxes or using OS acknowledgment systems.

3.4.3 Synchronization

Message passing can also be performed using synchronization which can be implemented either blocking or non-blocking model. The detailed description is presented as follows:

a) Blocking: The blocking method is synchronous (TCP) and can be implemented using blocking send or blocking receive.

Blocking Send: the sender will be blocked until the receiver receives the message (means the sender will not send the next message unless the receiver receives the previous messages).

Blocking Receive: The receiver will be blocked until a message is available (will not receive the other message).

b) Non-Blocking**:** The non-blocking method is asynchronous (UDP) and can also be implemented using non- blocking send or non-blocking receive.

Non-blocking Send: The Sender will send messages and continue to its next sending process (regardless of acknowledgement from the receiver).

Non-blocking Receive: The receiver will receive either a message or null from the sender and will continue whether it has received or not.

3.4.4 Buffering

In buffering, the communication between the two (2) processes takes place through a temporary queue. The message stores in a queue (when the sender sends the message) and then receiver picks the message from the queue. To implement the queue, we can use the any one of the three (3) methods:

a) Zero Capacity: The queue has no storage capacity, when the sender sends the message unless the receiver receives/picks the message, the sender cannot send another message.

b) Bounded Capacity: The buffer has limited queue of the messages and if the limit gets full and if the receiver is not picking up the messages then the sender will wait unless the buffer becomes available.

c) Unbounded Capacity: Unlimited number of buffers.

3.5 IPC Methods

There are three IPC methods through which processes share the memory and the resources:

- Sockets
- Remote Procedural Calls (RPC)
- Pipelining

3.5.1 Sockets

The socket is the endpoint of communication between the two (2) devices e.g. two (2) people (having mobile phones) want to communicate with each other. Then one of them initiates the communication (using mobile phone), the mobile phone acts as a socket (endpoint of communication).

The concatenation of the IP address and the port number is called socket. It defines the service and the IP address of the host system. For example, 192.168.2.1:80 (socket) tells the host name (192.168.2.1) and the service port (80, http). So, when two (2) computers communicate then in fact they communicate to the sockets (ports).

3.5.2 Remote Procedure Call (RPC)

In client-server model, the client requests for the services from the server. If the server is located outside the LAN then the server is called remote server. For example, if you want to get your clothes dry-cleaned then you need to call dry cleaner, the dry cleaner gets the service done and return you back. The server (dry-cleaner) gets the request done and returns the results to the client. In RPC, there are five (5) components:

a) *Client*: The user who requests for the services
b) *Client Stub*: The client manager which encapsulates and de-encapsulates the messages.
c) *RPC Runtime:* Transport layer (transportation time)
d) *Server Stub:* Encapsulates and de-encapsulates the messages.
e) *Server:* Execute the request

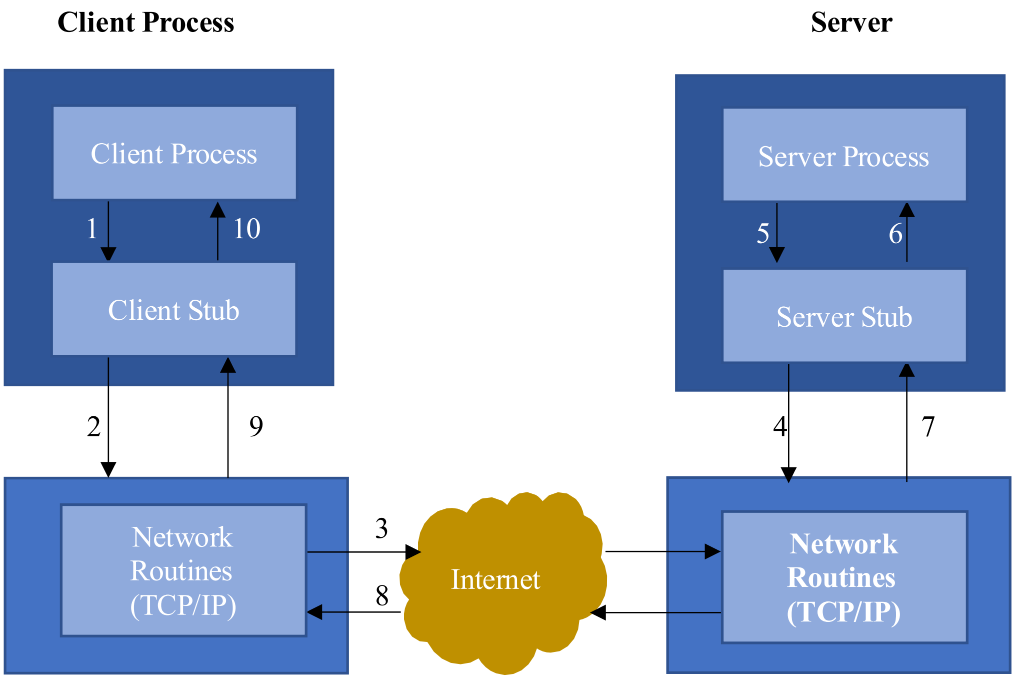


Figure 3.8: Remote Procedure Call (RPC)

The RPC steps are described as follows:

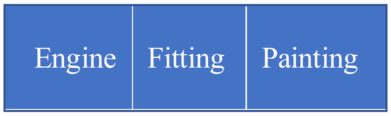
1) Client calls the client stub to forward the request.
2) Client stub builds the message and call the Local OS.
3) Client's OS sends the message to remote OS.
4) Remote OS gives message to the server stub.
5) Server stub unpacks the parameters and calls the server.
6) Server executes it and returns the results to the server stub.
7) Server stub packs the message and calls the local OS.

8) Server OS sends the message to the client’s OS.
9) Client’s OS gives the message to the client stub.
10) Client stub unpacks results and return to the client.

3.5.3 Pipelining

Usually, the OS performs the sequential execution on the processes. Let us consider the following example to better understand the prominence of pipeline.

We have auto manufacturing unit, where we have twenty-four (24) people work together to build a car. Assume that, there are only three (3) step process to build a car; Engine, Fitting and Paint.
In the sequential execution, all the twenty-four (24) people (altogether) do the engine job first, then fit the engine and finally paint the car. After finishing the one car, then they will work on the second car.



By using the sequential execution, only one (1) car can be built in one day. In order to improve the efficiency of work, we divide the people into three (3) group (G1, G2 and G3) of eight (8) employees. Now, first group (G1) builds the engine of car 1, handover it to fitting group (G2) and then start the engine job on car 2. Once, the G2 people fit the engine of car 1, they will move on to the car 2 and handover the car 1 to G3 (paint people). This mechanism is called the pipeline where we can improve the productivity and optimizes the hardware.

In computer (CPU) performs four actions on the processes:

- Fetch the instruction
- Decode the instruction
- Operand and variables
- Execute

The OS uses pipeline and divides the resources to perform these operations.
The pipelines can be of two types:

Ordinary Pipes: Pipes created by the parent process (cannot access outside the process).
Name Pipes: Can be accessed without parent-child relationship (between the different processes).

Chapter 4

Threads

Learning Outcomes:

- Overview of threads and its types
- Multithreaded Models
- Threading Issues

4.1 Overview of the threads

To implement the multiprocessing applications, the developers have mainly two options; Parent-child model and threading. Since, the threading is extremely lightweight (for processor) and fast therefore it is more common among developers than the than schemes. Each thread is responsible for only one single task, additionally, the thread creation is lightweight than the process creation.

4.1.1 Definition of the thread

The lightweight process which executes only one specific task or the basic unit of the CPU execution is called thread. One process can have many threads where each thread comprises of program counter, ID, set of registers and a stack (to store temporary variables). Some of the threads (Kernel Level Thread) also require TCB (Thread Control Block) which resides in Kernel. The threads share resources provided by process and cannot make system call, if it needs to access a resource then the thread requests through the process. The CPU executes one thread at a time, since, the context switching between the threads is very fast (apparently it seems simultaneous execution).

4.1.2 Benefits of threads

Some of the benefits of the threads are described as follows:

a) *Responsiveness*: Since, the context switching is extremely fast, so, if one thread blocks then CPU quickly moves to another thread.
b) *Resource Sharing:* Threads share resources of process which is much easier than Inter Process Communication (IPC).
c) *Economy:* No separate memory allocation for threads and no system call designs.
d) *Scalability:* Multi-core processors can be easily integrated in threading environment.

4.1.3 Multicore Programming

Typically, a CPU has two main components; Control Unit (CU) and Arithmetic Logic Unit (ALU). The CU fetches instructions and decodes the instructions while the ALU executes the instructions and forwards the results to I/O devices or RAM. The single core CPU contains one CU and one ALU while multicore CPU contains multiple CU and ALU (typically embedded onto single chip). These multicore CPUs execute can execute two tasks simultaneously. The multicore processors divide the activities, balance & data splitting, faster data testing and debugging. There are two ways to execute the tasks:

a) *Concurrency:* Running multiple tasks at a same time (practically one computation is running at a time).
b) *Parallelism:* The act of running multiple computations simultaneously (split the task between multiple resources).

In windows OS, by using Windows Management Instrumentation (WMIC) one can find the information of CPU and its cores (e.g. by using the command "CPU Get NumberOfCores, NumberOfLogicalProcessors).

4.2 Types of Threaded Processes

There are mainly two types of threaded processes; Single threaded and multi-threaded. The detailed description of these processes is presented as follows:

4.2.1 Single Threaded processes

The processes which include only one thread which performs one specific task. The figure 4.1 presents the single threaded process.

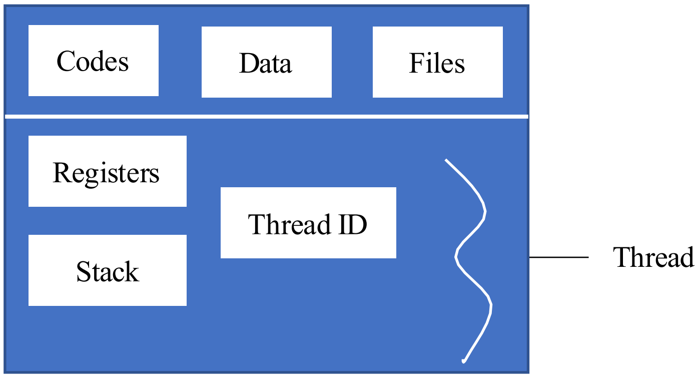


Figure 4.1: Single Threaded Process

4.2.2 Multi-threaded processes

In multi-threaded processes, each thread has its own ID, registers and stack, which they don't share with other threads. However, the codes, data and files offered by the process can be shared with other threads. The threads can create child threads but through processes. In multi-threaded processes, the threads are always co-operating (not independent) and cannot make system calls. The figure 4.2 presents the multi-threaded process concept.

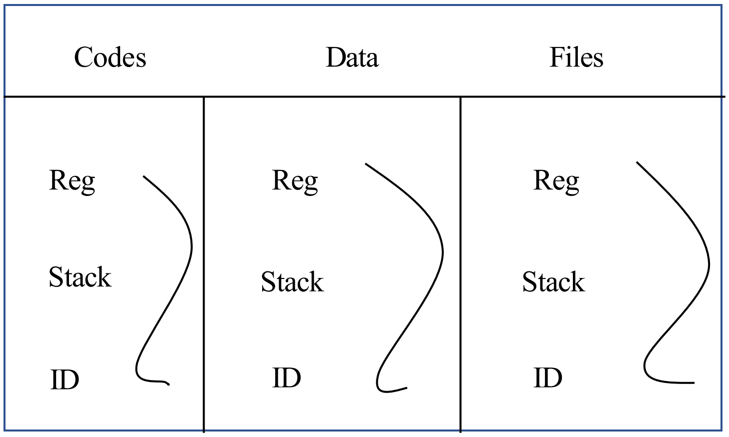


Figure 4.2: Multi-threaded Process

4.3 Types of threads

Now, if we are designing multi-process applications using multi-threaded processes then to create threads there are two options of the incorporated threads; User Level Thread (ULT) and Kernel Level Thread (KLT).

4.3.1 User Level Thread

The applications create threads using the thread library which exists within the user address space (e.g. in C programming ***pthread.h*** etc.). The thread library is responsible for:

- Creation and termination of the threads
- Communication between the threads
- Scheduling (writing, reading etc.)

For ULTs, the kernel is not aware of threading (doesn't know that the application has been designed using threads). The pros and cons of ULTs are summarized as follows:

Pros of ULT

1) Thread creation is simple and easy.
2) ULTs don't need any kernel level privilege and run on any OS.
3) ULTs are extremely fast and efficient.

Cons of ULT

1) Lack of concentration between threads and OS (Kernel) so, the process as a whole gets one time slice, irrespective of whether process has one (1) thread or one hundred (100) threads.
2) If one of the ULTs block then the entire process gets block.

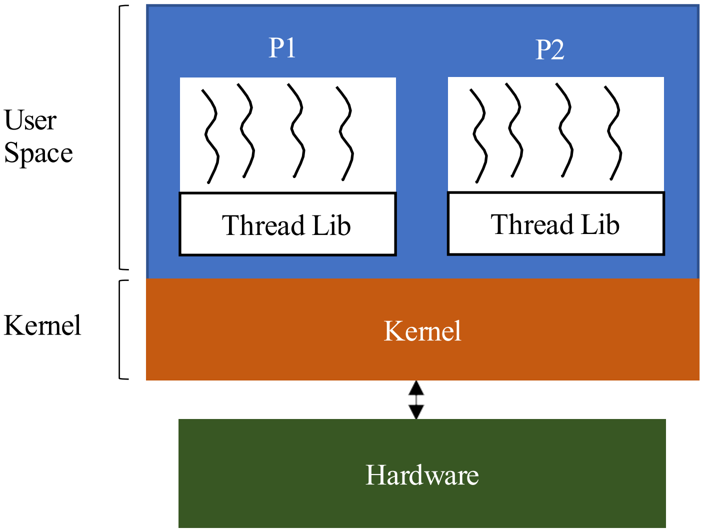


Figure 4.3: User Level Threads (ULTs)

4.3.2 Kernel Level Thread (KLT)

The KLTs are created and managed by the Kernel which means OS must be multi-threaded (supports hyper-threading). The Kernel allows the processes to create threads and do the thread management at Kernel (Thread libraries are located in Kernel). If a process has to create a thread then it has to make a system call. The pros and cons of KLTs are summarized as follows:

Pros of KLT

1) Multiple threads of same processes can be executed simultaneously and if one blocks then kernel execute/schedule the other threads.
2) More coordination between the threads.

Cons of KLT

1) Slower than ULTs (since system calls are involved).
2) Since, Kernel manages the threads then just like processes, it requires full TCB for each thread to maintain the information about the threads (more burden on RAM).

4.4 Multi-threaded Models

When a system supports both the ULTs and KLTs then we need multi-threaded models to map ULTs onto KLTs. There are three multi-threaded models:

- Many to one
- One to one
- Many to many

The detailed description of these models is presented as follows.

4.4.1 Many to One

As name suggests, many ULTs are mapped to a single KLT. The thread management is handled by thread library in user space. It doesn't support parallelism and only one thread (ULT) at a time can get kernel access. The Solaris uses many to one multithreaded design model in its design. The figure 4.4 depicts many to one model.

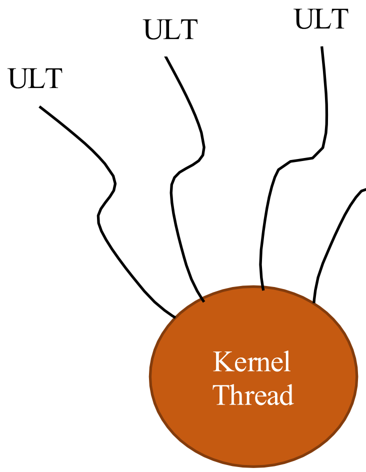


Figure 4.4: Many to One multithreaded model

4.4.2 One to One

One ULT is mapped to single KLT which means for each ULT, there is single KLT. If one ULT blocks then other threads continue to work. The only disadvantage of one to one model is each time whenever we create ULT then a corresponding KLT should be created (system call) for mapping which increases the overhead on OS.

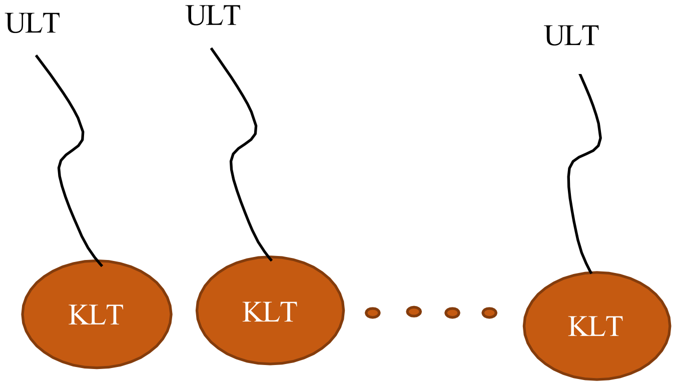


Figure 4.5: One to One multithreaded model

4.4.3 Many to Many

ULTs are mapped to equal number or a smaller number of KLTs. Usually, uniprocessors support smaller number of KLTs for ULTs while multiprocessors may offer equal numbers of KLTs. If one of the ULTs blocks then other can execute in parallel.

4.5 Threading Issues

In this section, we will discuss four issues of threading; Issues of system calls, thread cancellation, signal handling and thread local storage.

4.5.1 Fork( ) and exec ( ) system calls

For multithreaded systems, the interpretation of these system calls differs in its operation. The definition of these system calls is:

$fork(\,)$: Duplicating a process (create a child process) with different process ID.
$exec(\,)$: Replacing the content of one process with another but maintaining the same ID.

Let us understand the issues of fork ( ) system call, if one of threads makes fork ( ) system call then:

- Does new process duplicate all the threads? (some threads might be blocking)? or
- It will create a new process with single thread?

In Unix there are two versions of fork ( ) system calls; one duplicates all threads while the second type of system call only duplicates only that thread which involve that system call.
Similarly, does $exec(\,)$ system call replace all the contents of one thread (involved that system call) or whole process will get replaced? The answer is, it will replace all threads (entire process gets replaced).

4.5.2 Issues of signal handling

The major difference between the signal and the interrupt is the signal is event triggered by CPU and the software running on it while the interrupt is an event generated externally e.g. CPU is executing a process, (in the meantime) the user gives an input through the keyboard (I/O device) then an interrupt will be generated and the CPU suspends the on-going execution & get the interrupted task done first. The interrupt is asynchronous while the signal is treated as synchronous.
In Unix, the signal is used to notify the process that a particular event has occurred. So, the process can further proceed or terminate itself. There are two types of signal handlers:

a) *Default Signal Handler*: CPU defines how to handle the signal (predefined by the developer).
b) *User Defined Signal Handler:* User programs the process what to do, if any event like that happens.

The issue in signal handling is where the signal will be delivered? (thread who created that event or all threads). This problem can be resolved by using the thread pool which receives the signals and then based on instructions threads in thread pool terminate the specified thread or execute it further.

4.5.3 Thread Cancellation

Thread cancellation is the task of terminating a thread before it has completed. The threads need to be cancelled for several reasons e.g. many of the threads are searching for a specific item through a database and one of the threads returns the result then remaining threads might be cancelled.
The thread to be cancelled is called target thread. The cancellation of the thread may occur in two different scenarios.

a) *Asynchronous Cancellation:* One thread immediately terminates the target thread.
b) *Deferred Cancellation:* The target thread periodically checks whether it should terminate itself or not (allowing the thread an opportunity to terminate itself). The target thread will terminate itself whenever it will be safe to do so.

The thread cancellation can cause some resources reclaiming problem:

a) The resources have been allocated to the thread and it got cancelled. (how OS will reclaim the resources)? Usually, a thread is cancelled then OS reclaims all the system resources allocated to that thread. If the thread gets cancelled asynchronously then it would be difficult for OS to get all the resources back.

b) A thread is cancelled while in the middle of updating the data, it is sharing with other threads.

To avoid the thread cancellation, the deferred cancellation is the optimal technique where a thread checks itself and terminates when it will be safe to do so.

4.5.4 Thread Local Storage

As we know, threads share the resources provided by the process, so let say in a process we have a global variable G1, if any thread wants to execute something then firstly, it will load its variables into the local variables and then from local variables, the parameters will be loaded onto global variable of the process. The process makes the system call and forward the tasks to the kernel. Now, if the thread (T1) was loading its parameters onto G1 and at the same time T2 tries to load its values onto G1 then there could be a deadlock situation. This deadlock situation can be avoided with separate global variables for each thread. The figure 4.6 describes overview of thread local storage concept.

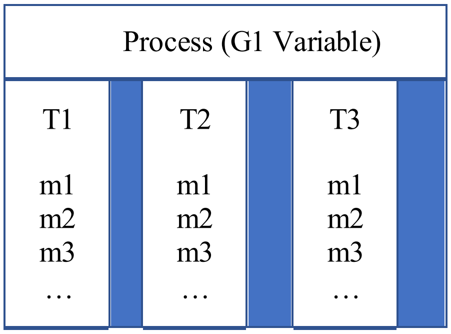


Figure 4.6: Thread Local Storage

4.6 Operating System Examples

In section, we will discuss the windows and Linux threads examples.

4.6.1 Windows Threads

The windows use one to one multithreaded mapping model where each thread has its own TCB (Thread Control Block). The TCB further includes EThread (Executive Thread), KThread (Kernel Thread) and TEB (Thread Environment Block) where TEB resides within the user address space while other two are located on Kernel. The figure 4.7 presents the structure of windows threads.

4.6.2 Linux Threads

The Linux threads are termed as tasks and just like windows the Linux OS also supports parallelism and multithreaded models.

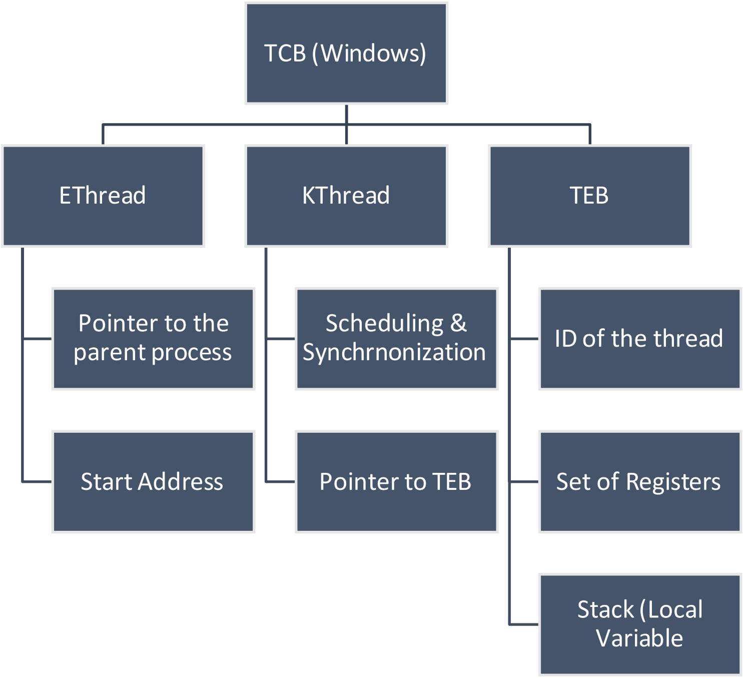


Figure 4.7: Thread Control Block of windows

Chapter 5

Deadlocks

Learning Outcomes:

- Deadlock Description
- Characteristics of Deadlocks
- Deadlock Handling

5.1 What is deadlock?

If a process is unable to change its waiting state indefinitely because the requested resources are held by another waiting process. For example, if you went to a party and you want to eat noodles. To eat noodles, you need two items; Plate and fork. Assume, that you got plate but couldn't find fork and there is another person who got the fork but no plate. Now, you are asking him to give you his fork and he is asking for your plate (no one is willing to step down). This scenario will result in deadlock situation. In multiprogramming environment, the number of processes execute simultaneously & compete for the limited number of resources. If a resource is not available then the process enters in a waiting state and if this waiting state depends on another waiting process then this situation creates a deadlock.

Let us understand the deadlock with another example. The figure 5.1 presents the deadlock concept.

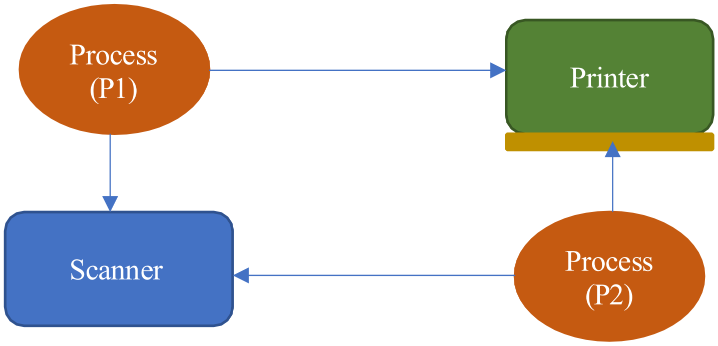


Figure 5.1: Deadlock Concept

For example, Process (P1) requires two resources (Printer and scanner) to execute itself. The P1 has already occupied scanner (runs the desired task) then request to access printer. In the meanwhile, another process (P2) (also requires printer and scanner) has already occupied printer (running its tasks) and requests for scanner. This situation leads towards deadlock where, P1 is requesting for printer (held by P2) and P2 is requesting for scanner (held by P1).

For the graphical view of the resources allocation process and visualizing the deadlock, the Resources Allocation Graph (RAG) is used. The RAG involves two attributes; Vertex (nodes) and Edge (connections/arrows). The processes are represented as circle and the resources are represented as square.

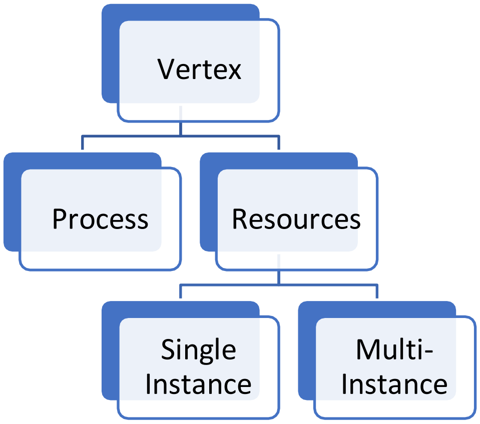


Figure 5.2: Vertex types

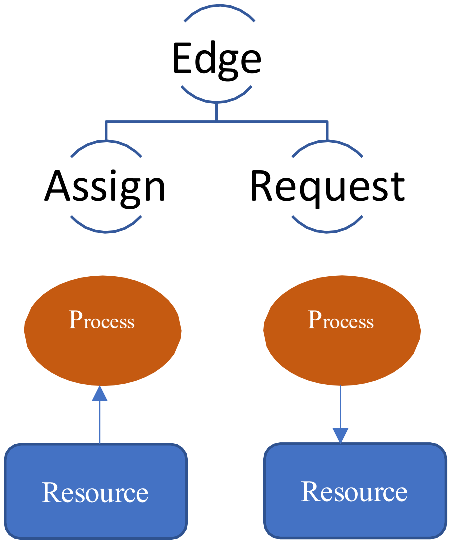


Figure 5.3: Edge types

If the edge (arrow) direction is from resources to process, it means resource is assigned to that process and if the direction of edge (arrow) is from process to resource then the process is requesting for resource. The following RAG represents the deadlock between two resources and the processes.

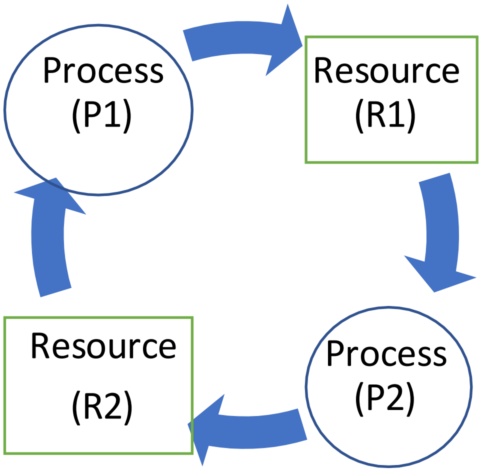


Figure 5.4: Resources Allocation Graph

In this example, both processes (P1 and P2) require two resources (R1 and R2) to get executed. The resource (R1) is assigned to P2 and resource (R2) is assigned to P1. The P1 is requesting for R1 (which is held by P2) and P2 is requesting for R2 (held by P1). Whenever the processes didn't get the requested resources then they enter in a waiting state. The waiting state can be finite (amount of time is known to the process, it may cause starvation) or infinite (amount of time is not known) where deadlock will surely occur. If the RAG doesn't contain cycle then there will be no deadlock. However, if the RAG contains a cycle with a single instance resource then deadlock will surely occur, while presence of multi-instance resource can avoid the deadlock.

Each process follows a system and involves three step model:

a) Every process requests for the resources.
b) If entertained then the process will use resource.
c) Process must release the resource after use.

5.2 Deadlock Characteristics

There are four (4) characteristics which result in deadlock.

- Mutual Exclusion
- Hold and Wait
- Non-Preemption
- Circular Wait

The detailed description of these four (4) characteristics are discussed as follows:

5.2.1 Mutual Exclusion

At least one resource type in the system should be non-sharing (Not sharable) and can be accessed one at a time (one by one). The sharing resources can be accessed by the multiple user at the same time e.g. watching a video on YouTube or Netflix etc. The deadlock occurs only if we have a resource which is mutually exclusive.

5.2.2 Hold and Wait

Recall the noodles example, where you acquire the resources (fork or plate) whichever was available and then start waiting for other resources.
A process is currently holding at least one resource and requesting for additional resources which are being held by other processes.

5.2.3 Non-Preemption

In the noodle example, you got a plate and there is another person who is holding the fork. You have snatched the fork him and ate the noodles. In this example, the deadlock will not occur. A resource cannot be pre-empted from a process by another process. The resources can be released voluntarily by the process holding it.
The deadlock will not occur unless all the four conditions meet (if above mentioned three characteristics occur and the circular wait is absent the deadlock will not occur).

5.2.4 Circular wait

Each process must be waiting for a resource which is being held by another process. The figure 5.5 presents the circular wait of the process.

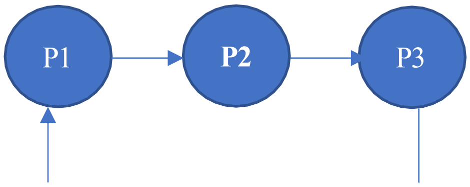


Figure 5.5: Circular wait of the Processes

5.3 Deadlock Handling

There are four methods to handle the deadlocks:

- Prevention
- Avoidance

- Detection and Recovery
- Ignorance

Prevention

The deadlock prevention methods are proactive approach where the system developers design an environment/system which ensures that deadlock should not occur. The prevention method involves the violation of deadlock four (4) conditions. The prevention methods are expensive and only used in real time systems.

Avoidance

System maintains a set of data (complete information of available resources, occupied resources etc.) and use that information to make the decision (whether it can entertain the process or not) to be in safe state.

Detection and Recovery

In the detection and recovery methods, we wait until the deadlock occurs and once we detect it then make the recovery plan.

Ignorance

The system developers ignore the problem as if it doesn't exist. The ignorance model is used if it is not worth it to solve the problem or not e.g. personal computers etc.

In this section, we will mainly discuss deadlock prevention method in detail. As discussed, the deadlock prevention is a high cost method which guarantees that deadlock will not occur.

5.3.1 Deadlock prevention by violating Mutual Exclusion

To avoid the mutual exclusion, we need to make all the resources sharable (multi-instances), which is practically infeasible.

5.3.2 Deadlock prevention by violating Hold & Wait

(a) *Conservative Approach*
The process is not allowed to acquire a resource unless all of the resources (require to execute it) are not available or in other words process is allowed to start execution if & only if it has all the required resources. This approach is extremely easy to implement but less efficient and not implementable on Unix and Linux. The process will not hold and wait the resources if all the required resources are not available.

(b) *Don't hold all resources together*
The process acquires a group of resources first then release the group & acquires the next group of resource.
For example, Process (P1) require ten (10) resources (R1, R2, …., R10) to get it executed. The process will acquire the resources GroupWise e.g. firstly, the process will acquire resources (R1-R4) and then access (R5-R10). Before each group request, the process has to release the previous group. This scheme is more efficient and implementable.

(c) *Wait Timeouts*
The process sets a wait time e.g. the process acquires the resources (whichever) are available and then held those resources for a specific wait time. If the remaining resources don't get available during that time then the process will release the held resources.

5.3.3 Deadlock prevention by violating Non-Preemption

If a process has a resource then another process cannot forcefully preempt that resource (only voluntarily releasing of resources). To avoid the deadlock and violate non-preemption condition, the system developers allows forceful preemption capability to high priority processes. The process which is in waiting state, will have less priority and must be selected as victim, while the process in running state is treated as high priority process.

5.3.4 Deadlock prevention by violating Circular Wait

To avoid the circular wait, rather than using need-based resources sequences, we can use increasing or decreasing order sequences for resources acquisition.
For example, Process (P1) needs resource (R2) first then R1. Similarly, the Process (P2) needs R1 first and then R2 to get it executed. The P1 tries to acquire R1, which is pre-acquired by the P2, so, P1 will not acquire any resource unless P2 finishes it execution. In other words, if the P2 occupies R1 first then P1 will be out of competition. The following steps are involved in violation of circular wait.

1. Assign numbers to all of the resources e.g. R1, R2, R3, R4, …., RN
2. Allows every process to acquire resources only in increasing or decreasing order e.g. P1 requires (R1, R6, R10) and P2 needs (R6, R9) then P1 first acquires R1 and P2 first acquires R6. Since, P2 has already acquired R6 so, P1 will be out of competition and release the held resources (R1). P1 will wait for the P2 to get executed and then start acquiring of the resources.

Since, there is no backward request is required therefore it will be in increasing flow and no possibility of the cycling.

5.4 Deadlock Avoidance and Detection

One of the most common method for deadlock detection and voidance is called Banker's Algorithm. In the banker's algorithm, we have to give detailed information to OS bout all incoming processes e.g. which processes are in running state, waiting state, holding time and need of resources etc. So, the OS decides which process can be executed or which cannot.
In banker's algorithm, the OS always makes a safe sequence (means no deadlock). To better understand the Banker's algorithm, let us consider the following example (Table 5.1). Some of the key terminologies used in the table are defined as follows:

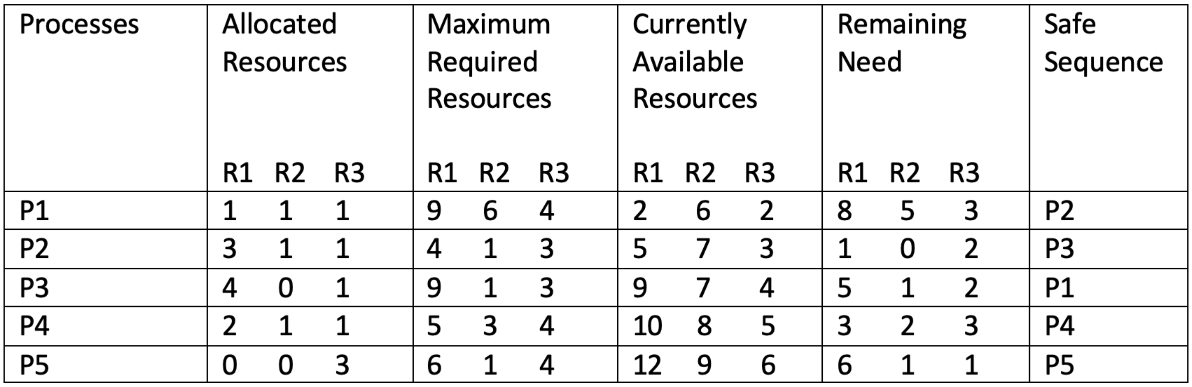
Allocation: Currently, how many resources have been allocated.
Maximum Need: The maximum requirement of the process to get it executed.
Currently Available: Currently available resources which can be calculated by using the following equation:

$$Current\ Available\ Resources = Max\ Need - Allocated\ Resources$$

If the $remaining\ need \leq Available$ then resources will be allocated and process will be executed and terminated. After the process gets terminated then it will release the held resources.

Note: Practically, the process cannot define the static need in advanced/dynamic need (based on the current computation).

Example: Assume that we have five (5) Processes (P1, P2, P3, P4 and P5) which require three (3) resources (R1, R2 and R3) to get it executed. If we have Eleven (11) instances of R1, seven (7) instances of R2 and ten (10) instances of R3. The table 5.1 presents the calculation of the safe sequence.



Processes	Allocated Resources			Maximum Required Resources			Currently Available Resources			Remaining Need			Safe Sequence
	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3	
P1	1	1	1	9	6	4	2	6	2	8	5	3	P2
P2	3	1	1	4	1	3	5	7	3	1	0	2	P3
P3	4	0	1	9	1	3	9	7	4	5	1	2	P1
P4	2	1	1	5	3	4	10	8	5	3	2	3	P4
P5	0	0	3	6	1	4	12	9	6	6	1	1	P5

Table 5.1: Banker’s Algorithm Example

Chapter 6

Memory Management

Learning Outcomes:

- Memory Hierarchy
- Memory Allocation Policies
- Address Translation and Paging

6.1 Basics of Memory Management

There are two (2) main components of the computer; CPU and Memory, which define what a computer can do or cannot do. We have discussed a lot about the CPU (scheduling, handling and deadlocks). Now, we will discuss the second component "Memory", its problems, issues and solutions.
To optimize the computational performance, there are three (3) important criteria:

Size: Large size memory is desirable so, it can handle/run large programs.
Access Time: The small access time is desirable, so, it should take less time to access data from memory. The machine will do things faster if the access time is small.
Per Unit Cost: The less cost is desirable.

However, all these three desires are contradicting e.g. Can we have a memory which should have larger size, small access time (faster) and less costly? If the memory size is large then it means we have more search space and it will take more time to search for a specific data.
"*Increasing the size of the memory will increase the access time and if we decrease the access time then we need to reduce the memory size*."
Therefore, it is practically impossible to have a single large memory with low cost and small access time.

6.2 Hierarchy of the Memory

To achieve large size, small access time and low cost, the concept of memory hierarchy is introduced. The figure 6.1 describes the concept of memory hierarchy.

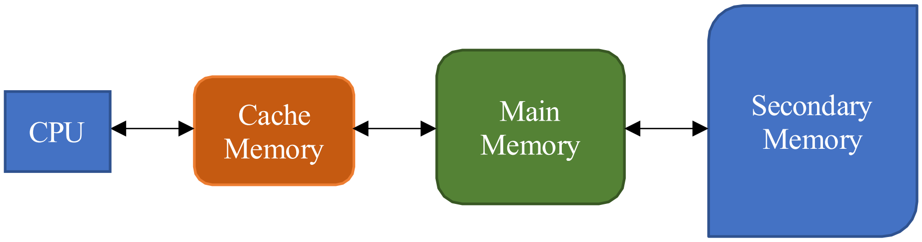


Figure 6.1: Memory Hierarchy

The memory hierarchy uses the "locality of reference" to offer three traits of the memory optimization. To understand the locality of reference concept, let us take an example of a process/program execution. As we know, the program executes (code) sequentially e.g. A program has one thousand (1000) instructions which are mainly stored onto Secondary memory (also called secondary storage). Currently, the CPU is executing instruction (I_{200}), then it will pre-fetch the next few instructions (e.g. I_{201} to I_{220}) to Cache Memory and some more instructions (e.g. I_{221} to I_{250}) which reduces the access time for CPU. So, the system doesn't need to access Secondary memory to access instructions of the process currently being executed. By using this concept, we can achieve the small access time and large storage as well.

We can calculate overhead access time of the CPU using hit ratio of the main memory. For example, hit ratio is 90% (data would be found in Main Memory) and the access time of the main memory is 10ms while the access time of the secondary memory is 100ms then:

$$\begin{aligned} Access\ time\ of\ the\ Process &= 0.9 \times (10) + 0.1\ (10 + 100) \\ &= 20\ ms \end{aligned}$$

There are two (2) important questions, which we will answer in this chapter:

1. How processes/instructions move from secondary memory to main memory which is called memory allocation (contiguous or non-contiguous)?
2. Whenever CPU generates address to fetch instructions (of currently being executed program), then it generates Logical Address (LA) which is understandable for the secondary memory but the main memory understands only Physical Address (PA). Since, the CPU mostly interacts with Main Memory (MM) (which needs PA) so, therefore address translation is required.

6.3 Memory Allocation Policies

There are two (2) memory allocation policies:

6.3.1 Contiguous Memory Allocation (CMA):

Move all the instances of the process and place all the instructions together.
Declaring an array is the best example of the CMA, when we declare an array in the memory then it allocates space in contiguous fashion (all together). The benefit of the contiguous memory is that, it offers small access time (faster) than other contending schemes. In array (contiguous), all the elements of the array are organized in a sequence and we can access any element directly. However, CMA creates external fragmentation where the memory gets saturated and doesn't allows instructions/processes to get stored onto main memory. Let us consider the following example to better understand the external fragmentation.
For example, we have a process, P1= 5KB which needs to be stored onto the memory of 10 KB (shown in figure 6.2). The 2KB of the memory is already been occupied by another process P2 and now we have 8 KB of the memory (two chunks of 4 KB). Although, we have 8KB of the memory but we need 5KB in a contiguous fashion which is not available in this case. This problem is called external fragmentation. The CMA always suffers from external fragmentation.

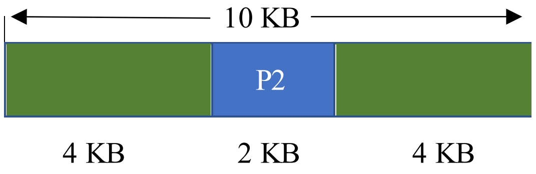


Figure 6.2: Contiguous Memory Allocation

6.3.2 Non-Contiguous Memory Allocation (N-CMA): In Non-Contiguous Memory Allocation (N-CMA), we break down the process instructions into small chunks and can be placed at different locations in Main Memory (MM). The best example of the N-CMA is linked list. In linked list, we use pointers to interconnect nodes, where first node knows the address of the second node and second node knows the address of third node and so on so far. The figure 6.3 presents the N-CMA example (linked list).

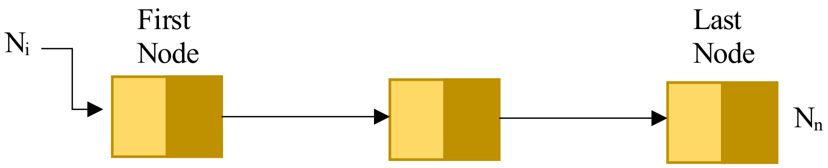


Figure 6.3: Non-Contiguous Memory Allocation (Linked List)

Now, if we want to access the last node (N_n), we cannot access it directly, it can be only accessed in linear fashion, which makes the system a bit slow. However, since we can place the process instructions in a non-contiguous fashion therefore, there will be no external fragmentation.

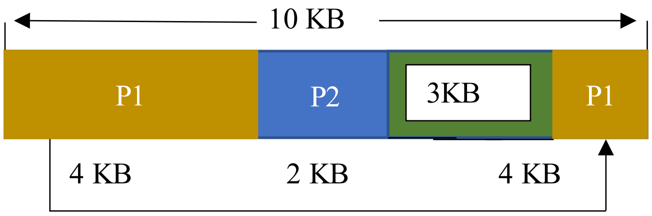


Figure 6.4: Example of Non-Contiguous Memory Allocation

6.4 Types of Contiguous Memory Allocation

In this section, we will discuss two methods to implement CMA; Fixed Sized Partitioning and Variable Sized Partitioning. The detailed description of both of these methods are presented as follows:

6.4.1 Fixed Sized Partitioning: As discussed earlier, in CMA, the process which moves from Secondary Memory (SM) to MM, it occupies the entire memory slice and all the instructions will be collocated. In fixed sized partitioning, the MM is divided into fixed sized partitions and the size of the partitions are not required to be of equal size. In fixed sized partitioning, beside the external fragmentation, the internal fragmentation may also occur. For example, we have a memory of 10KB (with four fixed sized partitions) shown in figure 6.5. Now, if we want to store a process of 4KB (P1) then we need to allocate the whole chunk to that process which cannot be reused (other processes cannot be stored onto that memory chunk). This will cause internal memory wastage which is called internal fragmentation.

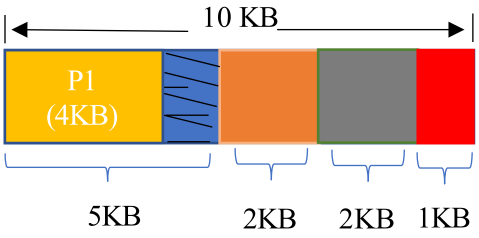


Figure 6.5: Example of Fixed Sized Contiguous Memory Allocation

The internal fragmentation occurs because the partition cannot be reused (one process/partition).

6.4.2 Variable Sized Partitioning: In variable sized partitioning, there is no pre-defined partitioning and the process can occupy the space based on its requirement. The variable size partitioning can never suffer from internal fragmentation; however, the external fragmentation can still occur. For example, we have a memory of 10 KB (variable sized) and we want to store two (2) processes (P1=4KB and P2=3KB). The figure 6.6 presents the example of the variable sized partitioning.

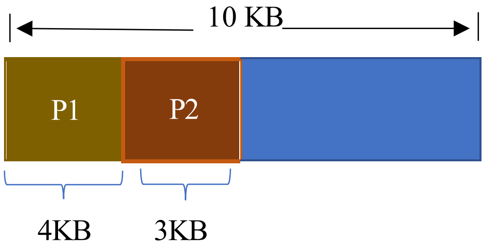


Figure 6.6: Example of Variable Sized Contiguous Memory Allocation

6.5 Contiguous Memory Allocation Algorithms

In this section, we will discuss three algorithms; first, best and worst with variable sized and fixed sized partitioning. The detailed description is presented as follows:

6.5.1 Variable Sized Partitioning: In this section, we will use first, best and worst algorithms to implement variable sized partitioning.

6.5.1.1 First Algorithm: The system starts searching for the space from the first available partition, once it finds the place the process/instructions will be stored onto it. e.g. we have a memory with three (3) Partitions (Pr); Pr_1=100KB, Pr_2=150KB, Pr_3=75 KB and if we want to store a process (P1=70 KB) then by using first algorithm, it will be mapped onto Pr_1.

6.5.1.2 Best Algorithm: The memory will be scanned and the system will search for the best match with the process (size) to be mapped e.g. we have a memory with three (3) Partitions (Pr); Pr_1=100KB, Pr_2=150KB, Pr_3=75 KB and if we want to store a process (P1=70 KB) then by using best algorithm, it will be mapped onto Pr_3.

6.5.1.3 Worst Algorithm: The memory will be scanned and the system will search for the worst match with the process (size) to be mapped e.g. we have a memory with three (3) Partitions (Pr); Pr_1=100KB, Pr_2=150KB, Pr_3=75 KB and if we want to store a process (P1=70 KB) then by using best algorithm, it will be mapped onto Pr_2.

To better understand the variable sized partitioning with these algorithms, let us consider the following example.

Example:
Assume that we have four (4) processes:
P1=300 KB
P2=25 KB
P3=125 KB
P4=50 KB
The current state of the memory is illustrated as follows:

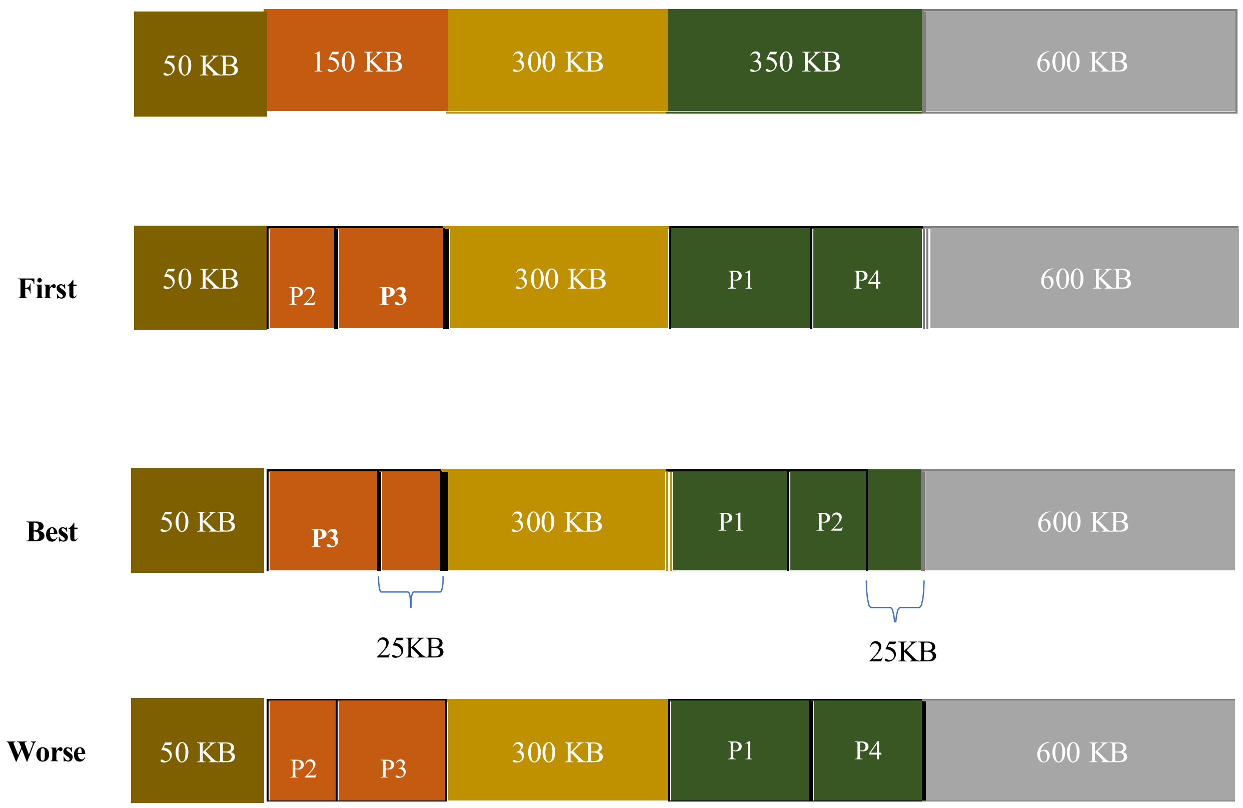


Figure 6.7: Example of Variable Sized Partitioning using first, best and worse algorithms

We can see from figure 6.7, by using first algorithm, to store the process (P1=300 KB), the first available memory slot is green (350 KB). Further P2 and P3 will be stored onto orange (150 KB) slot and finally the P4 will be stored onto the remaining green memory slot.

By using the "best" algorithm, the P1 and P2 will be stored onto green memory slot. The process P3 best fits with orange memory slot and after that there is no enough storage in contiguous fashion for P4 to be stored. So, it suffers from external fragmentation of 50 KB.

In "worse" algorithm, the P1 will be stored onto green memory slot (worse fit), P2 and P3 will be stored onto orange memory slot while P4 will be mapped onto the remaining green memory slot. There will be no external fragmentation in the worse algorithm.

6.5.2 Fixed Sized Partitioning:

In the fixed sized partitioning, besides the external fragmentation the internal fragmentation also occurs. As we know, in fixed sized partition, the size of each partition is fixed and only one process can be stored onto one partition. Let us use the three algorithms (first, best and worse) to understand the fixed sized partitioning.

Example:

Assume that we have four (4) processes:
P1=357 KB
P2=210 KB
P3=468 KB

P4=491 KB

The current state of the memory is illustrated in figure 6.8

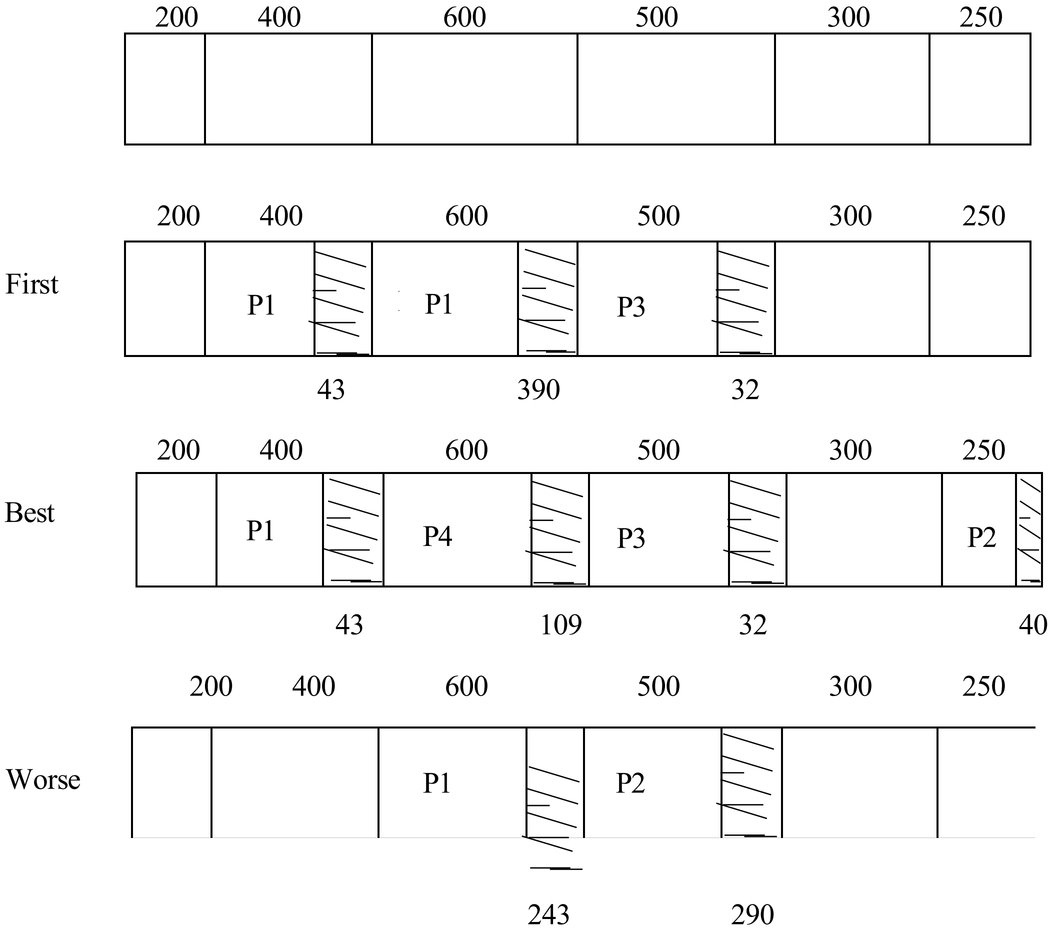


Figure 6.8: Example of Fixed Sized Partitioning using first, best and worse algorithms

- **Analysis of First Algorithm**:
    - Internal Fragmentation: $43 + 390 + 32 = 465KB$
    - External Fragmentation: 491 (P4 can be stored since, we don't have 491 KB in contiguous fashion).
- **Analysis of Best Algorithm**:
    - Internal Fragmentation: $43 + 109 + 32 + 40 = 224KB$
    - External Fragmentation: $0\ KB$
- **Analysis of Worse Algorithm**:
    - Internal Fragmentation: $243 + 290 = 533KB$
    - External Fragmentation: $468 + 491\ KB = 959\ KB$

We can clearly see that the best algorithm proves to be best in contiguous fixed sized partitioning.

6.6 Address Translation of Contiguous Memory Allocation

As discussed in section 6.2, the CPU generates the LA to fetch the instructions. The main memory only understands the PA; therefore, the LA needs to be converted to PA. The figure 6.9 presents address translation process.

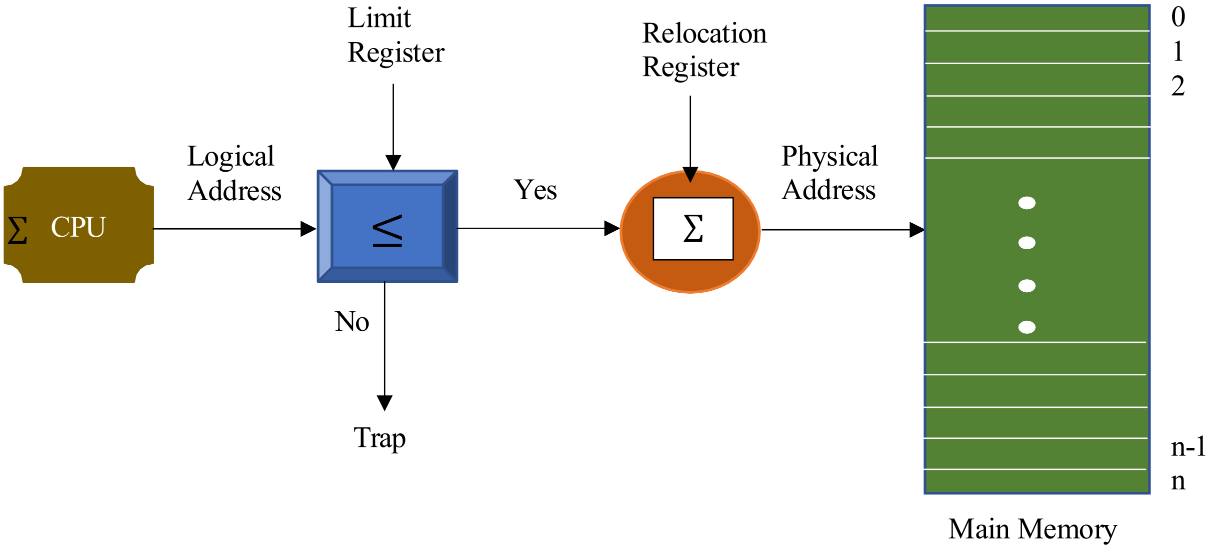


Figure 6.8: Logical Address translation for fixed sized Contiguous Memory Allocation

In contiguous memory allocation, the address translation (Logical to Physical address) is extremely easy. We need to create a data structure which holds the base address (limit register) (maximum number of registers in main memory for that process) of the process being executed. If the requested logical address is smaller than or equal (≤) to the limit register then it means it is a legit request and the relocation register (starting address of the process in main memory) will be added to the calculate the PA of the process. Let us understand the address translation with an example.
Assume that, a process (P1) is running on CPU and currently, the CPU is executing its instruction "Logical address=20" and CPU wants to fetch the next instruction "Logical address=21". If the maximum number of instructions (limit register) for this process are "100" then this request will pass the legitimacy test ($20 \leq 100$). Assume that relocation register (starting address of P1, I_1 (first instruction)) value is 500 then the physical address will be 521:

$$Physical\ Address = Logical\ Address + Relocation\ Register$$

If the CPU generates "Logical address=110" then this request will be turned down by the limit register, since, the $logical\ address \geq Limit\ Register$

6.7 Paging (Non-Contiguous Memory Allocation)

The N-CMA can be implemented using paging and segmentation. In this chapter, we will mainly discuss paging.
To avoid, the external fragmentation, we use N-CMA where, it is not mandatory to place all the instructions belong to specific process. For the N-CMA:

- Partitioning the Secondary and the Main Memory
- Size of each partition/block size is fixed (since, it is fixed therefore there would be some internal fragmentation but no external fragmentation.
- The partition of the secondary and the main memory size would be of the same size.

The partition of the secondary memory is called “page” while the partition of the main memory is called “frame”. The frames and the page are of equal sized to avoid misconfiguration when the instructions will move from secondary memory to main memory. The instructions of a process can be stored in a contiguous or non-contiguous fashion in secondary memory. Let us take an example to understand why N-CMA is different and difficult than CMA.
Assume, that we a process (X_1) whose instructions are stored over three pages (P_1, P_2 and P_3) (pages shown in figure 6.9 are shown in a contiguous fashion, which is not mandatory).

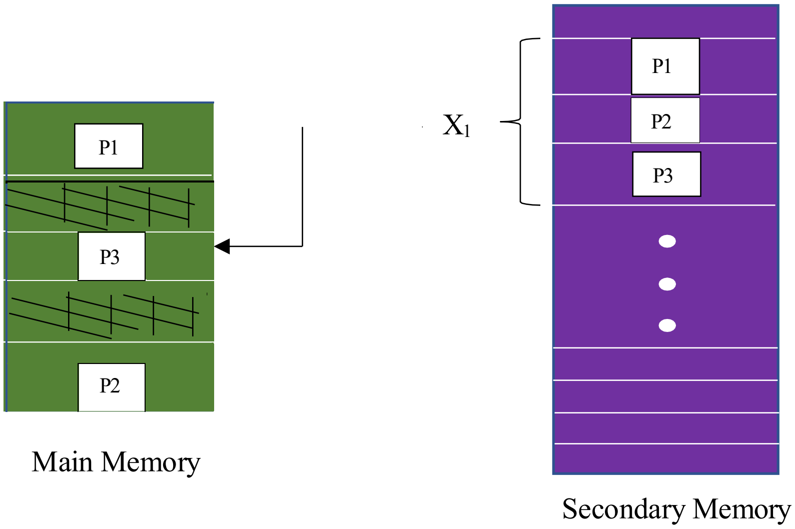


Figure 6.9: Non-Contiguous Memory Allocation Example

Now, we can see from the figure 6.9, the address translation method of contiguous memory allocation cannot be used since, the instructions are not collocated (sequential) on MM. In the CMA, we all need to know the Base Address (BA) of the process (on MM) and by adding the BA & the LA of the instruction, we can find the Physical Address of the process’s instruction. However, in N-CMA, the pages (instructions) are stored in a random fashion, therefore we need to find a different algorithmic solution to find the address of the pages (holding that instruction). In the N-CMA with paging, assume that the CPU wants to fetch the instruction 24 (I_{24}) on Page 2 (P2):

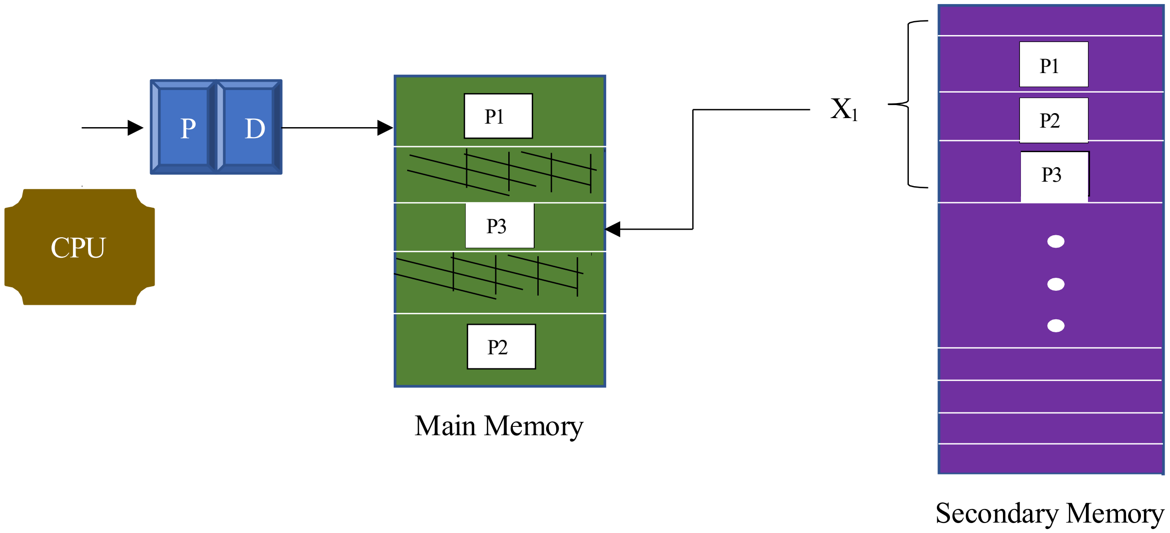


Figure 6.10: Non-Contiguous Memory Allocation Address generation

The Logical Address is divided into two parts P (Page Number) and the D (Instruction Offset). One of the methods of address translation (N-CMA) is using linked list, where the pages of the same process are interlinked with other pages using address pointers. The P1 knows the address of the P2 and P2 knows the address of the P3. All we need to know is the BA of the first page and then linked list will resolve the LA to PA. However, this method is extremely slow and makes the long overall access time.

6.7.1 Indexed Table:

In the indexed table approach, we maintain a data structure which is called Page Table (PT). The number of entries of PT are equal to number of pages of a specific process in SM. Each process will have its own PT, which contains indexes (base address of the frame of MM). To better understand the concept of indexed tables, let us consider the following example. The figure 6.11 describes the indexed table approach.
Assume that we have a process (X), which is stored over three (3) pages (P1, P2 and P3) onto SM. The CPU is currently running the process (X) and the all the three pages are moved from SM to MM in non-Contiguous fashion. The P1 stored onto frame (F4), P2 and P3 stored onto F2 & F5 respectively. The OS will create a PT for the process (X) which holds the frame address of the pages. Whenever the CPU generates the LA then PT finds the frame number and computes the PA. In the following example, the CPU wants to fetch the instruction 24 stored on P2, then:

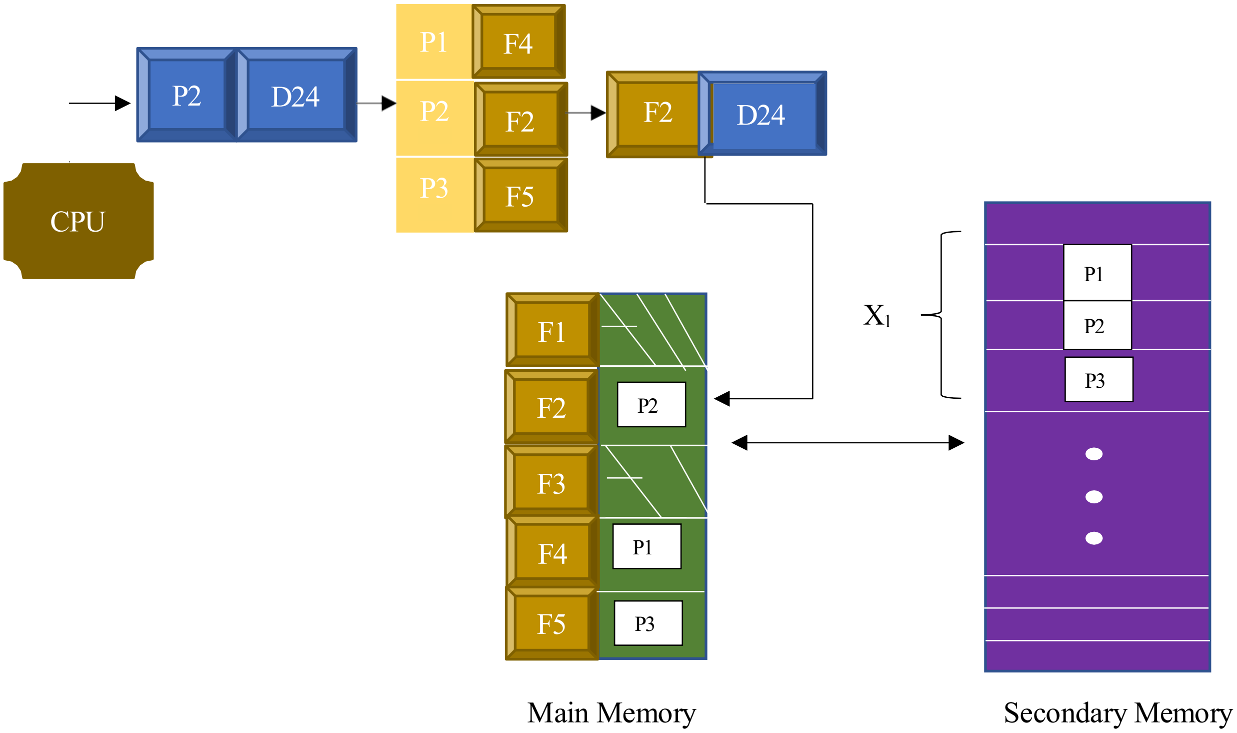


Figure 6.11: Non-Contiguous Memory Allocation using Indexed Table

- **Pros**:
  - Fast Access

    - No External Fragmentation

- **Cons:**
    - Overhead of PT for every process (not stored on PCB, it is stored separately).
    - Internal Fragmentation

Chapter 7

File System Interface

Learning Outcomes:

- File Access Methods
- Disk and Directory structure
- File mounting and sharing

7.1 File Concept

A file is a collection of bits, bytes and characters. The computers understand/interpret the data in the form of files e.g. xyz.mp3 etc. The file can be defined as "*Named collection of related information on secondary storage.*" The data cannot write to secondary storage unless it is in a file format where each file format/system has a different structure e.g. .txt, .mp3, .bat and. docs etc.
There are mainly three (3) types of files:

Text Files: Sequence of characters organized into lines.

Source Files: Sequence of sub-routines or functions (executable to perform the tasks)

Object Files: The collection of words/sequences of bytes organized into blocks, when we run a program which uses the digital resources, it mainly utilizes the objects files for transferring of the instructions.

The files can be divided/differentiated based on the type of the data e.g. Numeric, characters and the binary, all the file related parameters are contiguous.

7.1.1 File Attributes

The file attributes describe the properties files, which defines type and name which are used to track the file within OS.

Name: Human readable form (file name).
Identifier: Unique identifier (tag) (Number assigned to a file by the OS).
Type: File type (text, mp3 or source file etc.)
Location: Pointer to a device and the location of the file (where the file is stored).
Size: Current size of the file.
Protection: Access control information of the file e.g. who can read, write and execute.
Time and Date: To maintain the logs (when the file was created and modified (metadata)).
User ID: Information of the file creator.

7.1.2 File Operations

There are five (5) main operations, which OS should be able to perform on the files.

- **Creating a file**: The OS should find the space in the system for the file and enter the specified filing system.
- **Writing a file:** OS makes system call to write onto the file.

- **Reading a file:** To read a file, OS makes a system call.
- **Repositioning a file:** Allows the positioning the file cursor onto the specific byte, in order to start reading from the point ahead (file seek).
- **Deleting a file:** Delete the file entry from the file system.
- **Truncating a file:** Deleting the contents of the file without updating the file attributes.

7.2 File Access Methods

There are three (3) file access methods; Sequential, direct and indexed access methods.

7.2.1 Sequential Access

In the sequential access, the records will be accessed in a proper sequence (as they will be printed out or output). One record is processed at a time:

- **Read Next**: Read the record and move the pointer to the next record.
- **Write Next**: Write the record and move the pointer to the next record.
- **Rewind**: If you want to go back (moving back).
- **Skip:** It allows the user to skip 'n' records

The figure 7.1 describes the concepts of rewind, read/write and skip operations.

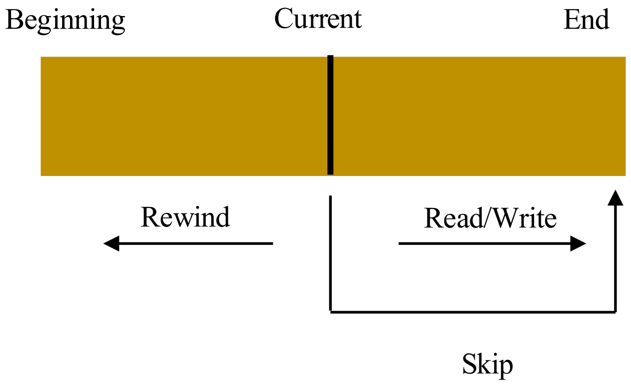


Figure 7.1: Sequential Access Method

7.2.2 Direct Access

The direct access method is called disk model, which allows users to jump to any record (as per user's need). In direct access OS supports following three (3) operations:

Read 'n': Reading record number 'n'
Write 'n': Writing the record number 'n'
Jump 'n': Jump to instruction 'n'

7.2.3 Indexed Access

In the indexed access methods, we organize the files using tables; indexed files, sub-index files and relative files. The file is stored onto the relative file and pointers of the indexed files link the file with the relative file.

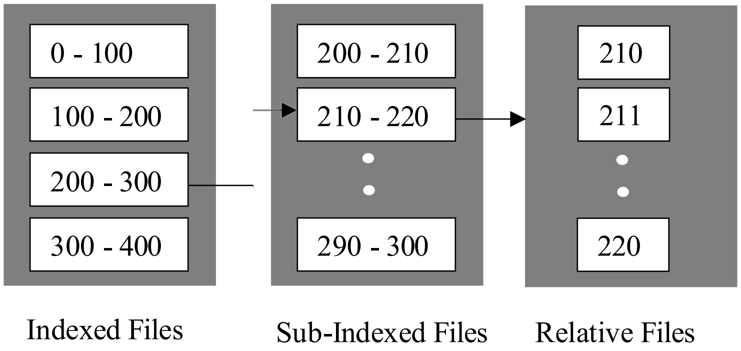


Figure 7.2: Indexed Access Method

7.3 Directory Structure

The directories are used to organize the files for efficiency and for user(s) convenience. By using the directories, we can store the files and their attributes. The file system allows following operations in directories:

- Search a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse a file system

There are four (4) directory structure models; Single level directory, Two level directory, Tree structure and the acyclic graph directory. The detailed description of the directory structure models is discussed as follows:

7.3.1 Single Level Directory

A single directory for all users, who can create and manage the files in that particular directory.

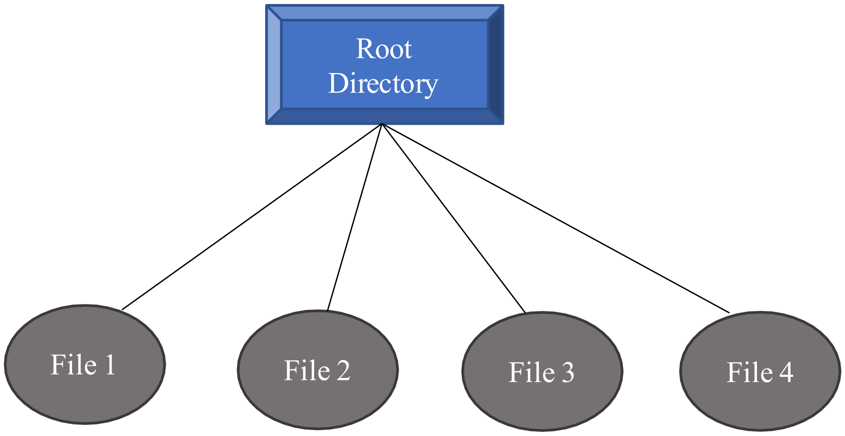


Figure 7.3: Single Level Directory

The single level directory is simple to implement, however it offers naming and grouping problems (The files belong to same application can get overlapped).

7.3.2 Two Level Directory

It offers separate directory for each user; therefore, the different users/applications can have same name files which improves the searching of the files. However, it still prone to naming problem. The figure 7.4 presents the two-level directory structure.

7.3.3 Tree Structure Directory

The tree structure offers user defined various levels (sub directories) as per user's need. It resolves the naming problems and improves the file searching, however, the sharing of the files between the directories are still didn't get resolved.

The figure 7.5 depicts the concept of the tree structure directory.

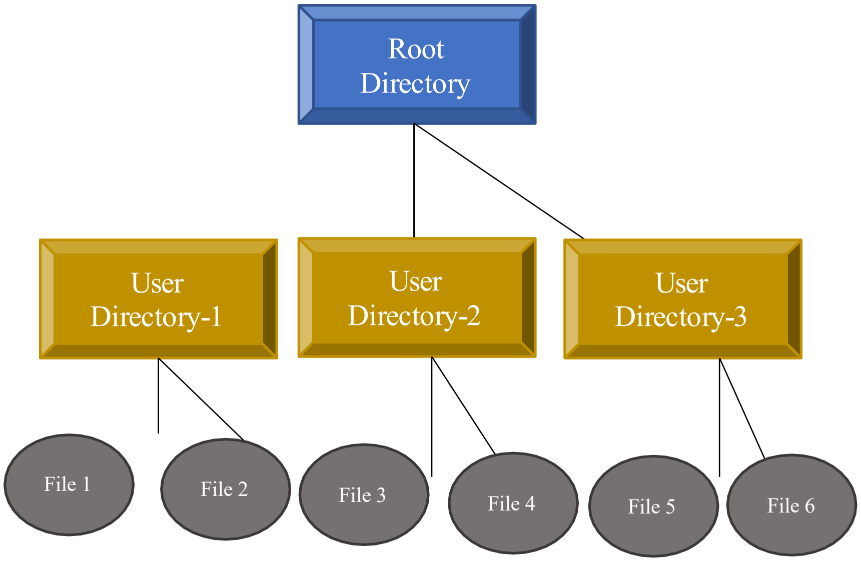


Figure 7.4: Two Level Directory

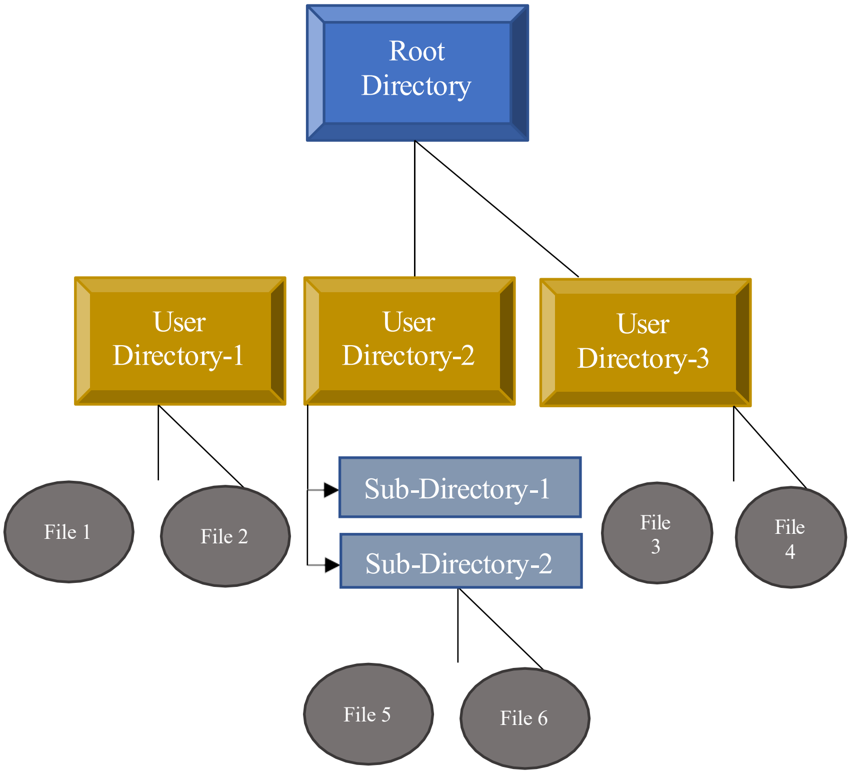


Figure 7.5: Tree Structure Directory

7.3.4 Tree Structure Directory

The tree structure allows the sharing of files between the different directories and offers all those six (6) operations of the file system.

7.4 File system mounting

Mounting is the process of attaching files to a specific file system. The file system mounting makes the OS files and directories available at a specific path for accessing. As we know, usually the OS and program files are stored onto C drive, while the D & E partition holds the user's data. For example, we have installed software "Eclipse" (JAVA IDE), the source files of the software will be located in C drive (C:\Eclipse). Now, if we have designed an application e.g. Calculator which requires JAVA to run the application and the file needs to be mounted with the source files of JAVA application. To mount the file with the source file, the following command is used:

$$D{:}\backslash mount \quad C{:}\backslash Eclipse\ \ D{:}$$

This command allows the eclipse files to be used in D drive. When we uninstall the application then we need to detach the files from the file systems (Unmount):

$$D{:}\backslash unmount \quad C{:}\backslash Eclipse\ \ D{:}$$

7.5 File system sharing

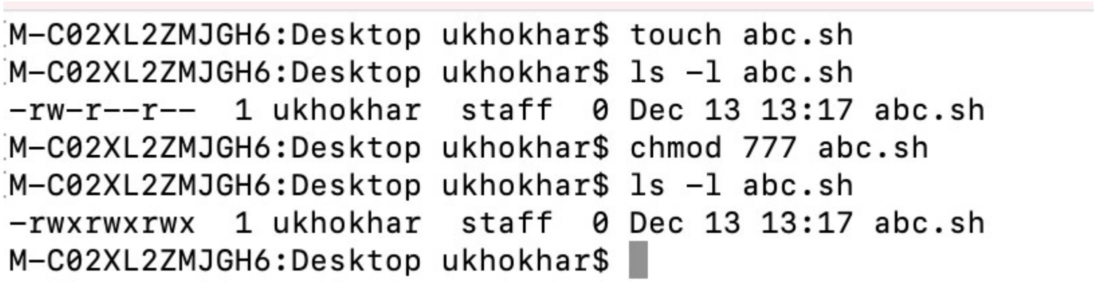
The file sharing allows the multiple users to access/share the files. When we enable the file sharing then the access control and protection method must be configured to protect the files from unauthorized access. In *nix systems, the users are categorized in three types; Administrators, Groups and other (public) and users can have three (3) access rights; Read (R), Write (W) and Execute (X). The administrators can create and manage the users, assign them a specific group with certain rights or they can have general access of the system. The users having full privilege (RWX) are assigned '7' (means user can read (4), write (2) and execute (1)). If the administrators want the users to only execute the file then '1' will be assigned and similarly '2' allows the users to have write privilege. For example, we create a file (abc.sh) in a Linux OS then to find its default access privileges, the following command is used:

$$ls\ -l\ \ abc.sh$$

It will show the permissions of the file abc.sh for all the three types of user. If the administrator wants all the three types of users to have full privilege then:

$$chmod\ \ 111\ \ abc.sh$$

The figure 7.6 shows the permissions of the file:



```
M-C02XL2ZMJGH6:Desktop ukhokhar$ touch abc.sh
M-C02XL2ZMJGH6:Desktop ukhokhar$ ls -l abc.sh
-rw-r--r--  1 ukhokhar  staff  0 Dec 13 13:17 abc.sh
M-C02XL2ZMJGH6:Desktop ukhokhar$ chmod 777 abc.sh
M-C02XL2ZMJGH6:Desktop ukhokhar$ ls -l abc.sh
-rwxrwxrwx  1 ukhokhar  staff  0 Dec 13 13:17 abc.sh
M-C02XL2ZMJGH6:Desktop ukhokhar$ 
```

Figure 7.6: Permissions of the users in *nix OS

Each user is assigned a specific user ID (to attach the permission with the user) and Group ID (defines the set of permissions attached to that group).

Chapter 8

Cloud Computing Concepts

Learning Outcomes:

- Cloud Computing Introduction
- Cloud Computing Service Models
- Cloud Computing Technologies

8.1 Cloud Computing Introduction

The "cloud" refers to servers that are accessed over the internet, and the software and applications that run on these servers collectively is called cloud computing. Cloud servers are located in data centers all over the world and are usually undisclosed for security reasons. By using cloud computing, users and companies do not have to manage the physical servers themselves or run the software application on their own machines.

The cloud enables users to access the same files and applications from nearly any device that has a network connection and the processing occurs on the remote servers in the data centers instead of on the local user device. This how webmail application providers such as Gmail and Microsoft 365 as well as cloud storage providers like Google Drive and Dropbox have always operated. However more recently, cloud computing expands this concept to apply it to emerging technologies such as Computing, Database processing, Machine Learning and other new innovations.

Switching to cloud computing lowers IT costs and overhead allowing for smaller business that may not have the capital for their own data centers allowing them to be more competitive. The cloud also makes it easier for organizations to operate internationally due to the remote locations of these servers that are all in-synch with each other allowing access from anywhere worldwide. The following section will discuss the four types of cloud deployments:

8.1.1 Private Cloud

A private cloud are many servers located in one or more data centers that are distributed over a network dedicated to one organization and its subsidiaries. Private clouds permit only authorized users, providing the organization greater control over data and its security. Business organizations that have dynamic, critical, secured, management demand could look to implement a private cloud.

Advantages

- Highly private and secure
- Control oriented allowing for more control over its resources from both accessibility and security standpoints

Disadvantages

- Lack of scalability since it is limited to internal host resources
- More expensive since it offers more security and features
- Pricing is costly as the organization is responsible for all capital expenditures
- Restrictive to local access so is difficult to expand globally

Private clouds have been used for businesses for quite some time and is not a new concept which contrasts with the next deployment.

8.1.2 Public Cloud

A public cloud is a service run by an external vendor and that vendor hosts the servers in one or more of their data centers.

Unlike a private cloud, public clouds are then shared with other organizations even though the individual organizations may not be aware or even be concerned with this fact.

The servers use virtual machines created on the physical serves and are made available to whoever wants to pay for the usage of that virtual machine. This is an example of multitenancy allowing for multiple tenants to essentially rent the same server processing capabilities.

Server infrastructure belongs to the cloud service providers (CSPs) that manage them and control the resources allowing for companies to not worry about capital expenditures. Public clouds have become more and more popular due to increased network connectivity and as well as the desire to lower IT operating costs.

Advantages

- Highly flexible and reliable
- Highly scalable
- Low startup costs and relatively lower operating costs
- Geographical independence

Disadvantages

- Less secure since your data is no longer exclusively in your possession
- Risk of vendor lock-in in case you wanted to change CSPs

8.1.3 Hybrid Cloud

Hybrid cloud implementation combine both public and private clouds and may also include on-premise servers. An organization may use the private cloud for some more secure applications and then the public cloud for other less secure or stringent applications. Having both public and private has its benefits such as backup and redundancy but can also lead to higher operating costs.

Hybrid cloud deployment not only safeguards against vendor lock-in but allows for more selective use of both the private and public cloud deployment models leading to maximizing advantages and minimizing disadvantages.

Advantages

- Flexible and reliable
- Secure
- Cost effective
- Scalability as needed

Disadvantages

- Complex networking required
- Security and compliance challenges
- Interoperability of various cloud implementations

8.1.4 Community Cloud

Community cloud implementation shares infrastructure between several organizations from a specific industry or community with common concerns such as security, compliance and jurisdiction. These implementations could be managed internally by the organizations or by a third-party and the costs are spread over the users allowing for potential savings of the entire community.

For joint business organizations, ventures, research organizations and the like a community cloud can be the appropriate solution. Community cloud members need to collaborate and evaluate which type of cloud hosting is best for the community and decide on who will manage it.

Advantages

- Cost reduction
- Increased security, privacy and reliability
- Ease of sharing and collaborate
- Potential compliance benefits

Disadvantages

- Higher cost compared to the public deployment model
- Sharing of fixed compute and storage resources
- Management concerns
- Not as popular solution

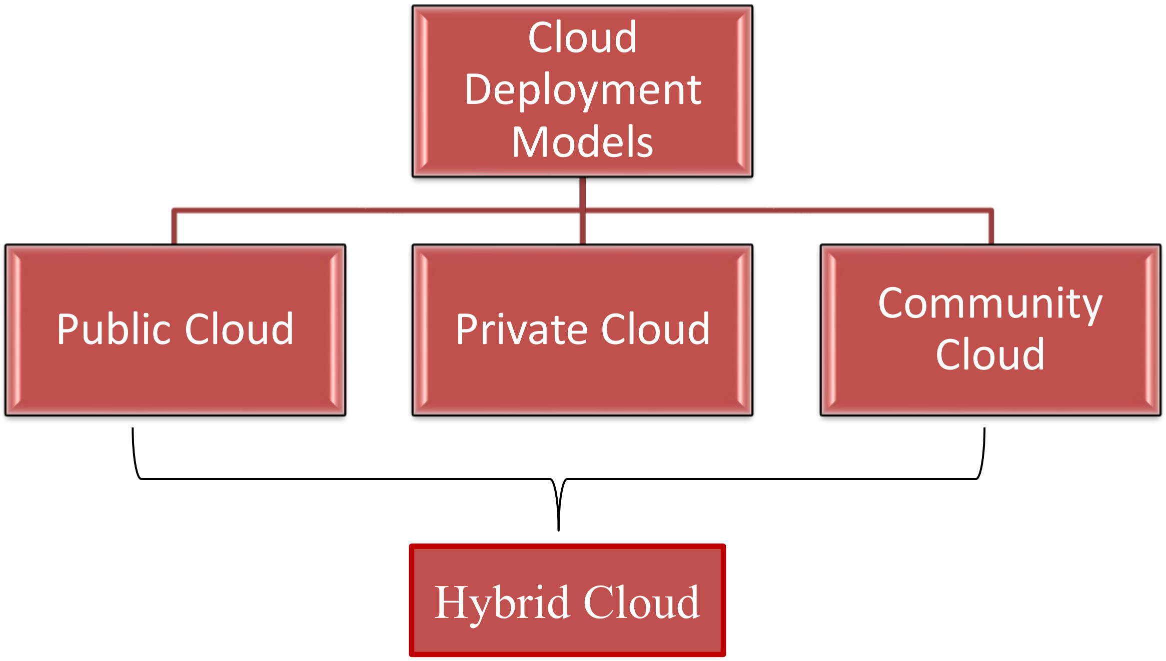


Figure 8.1: Types of Cloud Models

8.2 Cloud Computing Service Models

Once a cloud deployment model has been determined another decision is to decide on the three main cloud service models to satisfy the unique set of business requirements. These three models are known as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS).

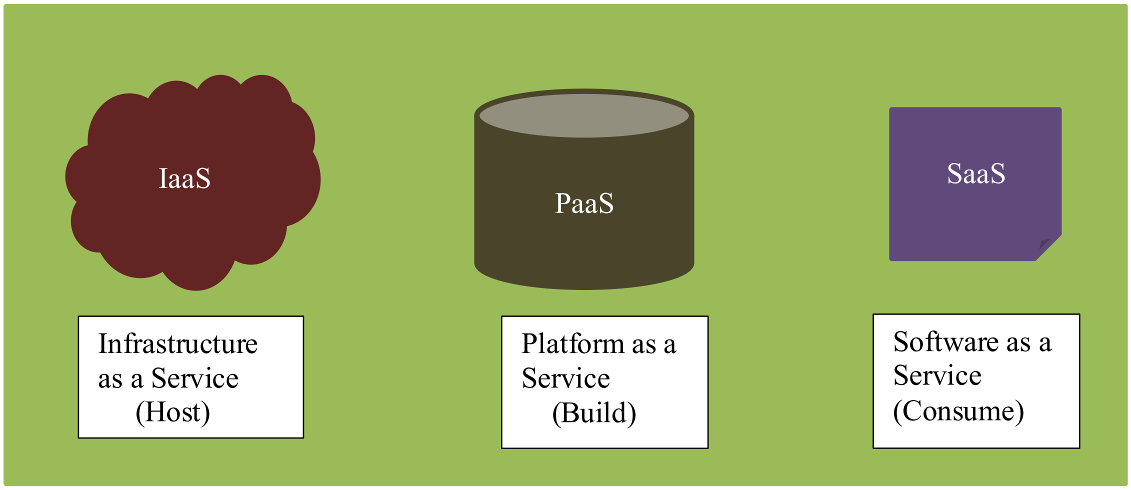

IaaS
PaaS
SaaS
Infrastructure as a Service (Host)
Platform as a Service (Build)
Software as a Service (Consume)

Figure 8.1: Models of Cloud Services

8.2.1 Infrastructure as a Service (IaaS)

Infrastructure as a Service (IaaS) is a self-service model for managing remote data center resources. IaaS provides for virtualized computing resources over the Internet hosted by the third-party cloud service provider such as Amazon Web Services, Microsoft Azure, or Google Cloud.

Instead of an organization having to spend vast capital on purchasing hardware, they pay to use IaaS using the consumption model effectively converting capital expenditures to operating expenditures. This allows for much more flexibility and cost savings thus leading to is popularity.

Most organizations use IaaS since they are already a familiar with IaaS when they managed and maintained their on-premise infrastructure but IaaS allows for much more flexibility and is highly scalable.

Effectively companies are renting the servers and storage they need from the cloud service provider but they need to provide their own knowledge and technical expertise to maintain and manage them. Examples of IaaS include Amazon Elastic Compute Cloud (EC2), Digital Ocean, RackSpace and many others.

8.2.2 Platform as a Service (PaaS)

Platform as a Service (PaaS) allows organizations to build, run and manage their applications without needing to worry about IT infrastructure. This allows them to easily and rapidly develop, test and deploy applications.

Developers can focus on writing code and creating applications without doing time-consuming IT infrastructure activities such as provisioning of servers, storage, backup and configuration.

PaaS brings more value to cloud. It reduces management overhead thus lowering costs allowing organization to focus on revenue generating activities include of infrastructure upkeep and maintenance.

PaaS can be compared to renting all the tools and equipment necessary for building a house instead of renting the house itself. Examples of PaaS include: Amazon Elastic Beanstalk, Heroku and Microsoft Azure App.

8.2.3 Software as a Service (SaaS)

Software as a service (SaaS) replaces the traditional on-device software with software that is licensed on a subscription basis. The software is centrally hosted in the cloud by the cloud service provider. Most SaaS applications can be easily accessed directly from a web browser or app without any downloads or installations though some may require plugins.

Users do not necessarily need to install applications on their device they just need to have Internet connectivity to access them. SaaS is like renting a house where the landlord maintains the house, but the tenant gets most of the use of it as if they owned it. Examples include: Salesforce, MailChimp, Slack, Webmail and numerous other popular applications.

In summary with Infrastructure as a Service the users host their applications. Platform as a Service allows for the users to build their applications whereas Software as a Service allows them to use or consume the applications.

8.3 Cloud Computing Technologies

Cloud computing has enabled many new technologies such as Machine Learning, Artificial Intelligence systems, Data Analytics, Big Data, Data-warehousing and many more. However, the underlying technology that allows for Cloud computing to be so popular is virtualization. The next section briefly introduces virtualization and its many benefits.

8.3.1 Virtualization

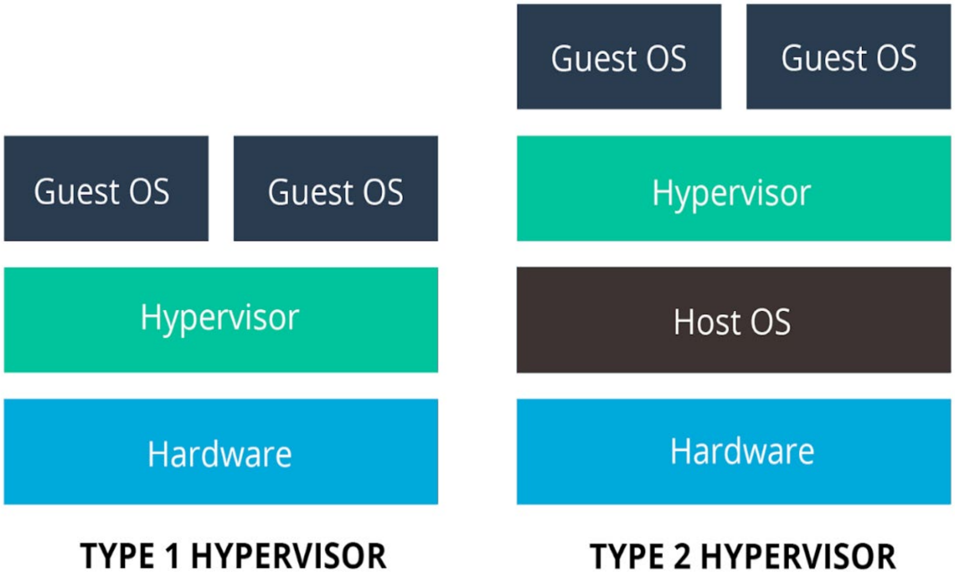
Virtualization is the act of creating a virtual rather than a physical version of a computing entity such as computer hardware, computer processing, storage or computer network resources. Virtualization began in the 1960s when used by mainframe computers between different applications but the technology has become much more popular in the 2000s as personal computing hardware became more powerful.

Virtualization is the process of running a virtual instance of a computer system in an abstraction layer on top of the physical hardware. More commonly, it refers to running multiple operating systems on a computer system simultaneously. The applications running on the virtualized machine appear as if they are running on their own dedicated machine and unconnected to the host operating system which sits below it.

There are many reasons why organization utilize virtualization in computing. For example, desktop users can share common applications that are written for different operating systems without having to switch or reboot into a different operation system. A very common case is when an application runs on both a Microsoft Windows and an Apple Mac system. If they are both virtualized the applications are accessed by the end user independent of the underlying operating system. For server administrators, virtualization offers the ability to run many operating systems on one physical server by creating virtual machines allowing more efficient use of the physical hardware while isolating various applications for efficiency and security. This is essentially how public cloud computing evolved.

To be able to perform virtualization the abstraction program is called a hypervisor and are traditionally categorized into two classes: Type I or "bare metal" hypervisors that run guest virtual machines directly on a computer system's hardware effectively behaving as an operating system. Examples of Type I hypervisors include Microsoft Hyper-V and VMware ESXI.

Type II, or "hosted" hypervisors behave more like a traditional application that can be started and stopped by a normal program. Examples of Type II hypervisors include VMWare Workstation and Oracle's VirtualBox.



The virtual machines that are created emulate the computer system sharing the physical system hardware and thus have access to the host machine's CPU, memory, one or more virtual or physical disk devices; one or more virtual or physical network interfaces as well as other devices such as video cards, USB devices and more.

There are numerous benefits from using virtual machines and some of them include:

- Lowing IT infrastructure expenses
- Maximizing current capital expenditures
- Increased efficiency and productivity

- Reducing downtime and enhancing resiliency in disaster recovery situations
- Control independence from SysOps and DevOps
- Move to more green-friendly environments

As one can see these benefits are also benefits seen from using cloud computing; thus, virtualization is the seed that revolutionized the computing world leading to cloud computing.

8.3.2 Containers

Since the benefits of virtualization and virtual machines are clearly established and have been implemented by many organizations the next evolution of virtual machines is something called a container which will be briefly introduced in this section.

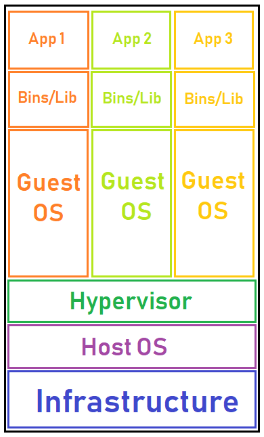
Containers are packages of software that contain all of the necessary elements to run in any environment. In this way, containers effectively "virtualize" the operating system and an be ran anywhere from a private data center, pubic data center, public cloud or even an individual personal laptop or mobile device.

Containers allow for development teams to move fast, deploy software applications efficiently and operate at unprecedented scale. Benefits of containers include the separation of responsibility, workload portability and application isolation among others.

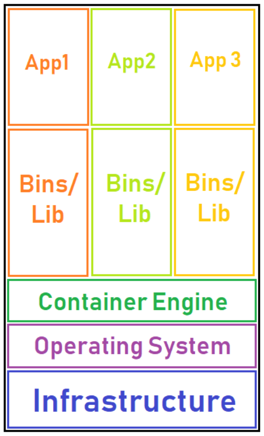
Containers are often compared to virtual machines (VMs) but are far more lightweight and carry these advantages:

- Containers are much more lightweight than VMs
- Containers virtualize at the OS level while VMs virtualize at the hardware level
- Containers share the OS kernel and use a fraction of the memory VMs require

Consider the physical containers on transport ship where the ship itself is the virtual machine carrying many containers which are individually isolated and independent.



Virtualization



Containerization