Chapter 1: Introduction to Operating Systems

Sukomal Pal, CSE, IIT(BHU)

Chapter 1. Introduction

- Concept of Operating Systems
- Generations of Operating systems
- Types of Operating Systems, OS Services, System Calls
- Structure of an OS - Layered, Monolithic, Microkernel Operating Systems
- Concept of Virtual Machines
- Case study on UNIX and WINDOWS Operating System.

COMPUTER FUNDAMENTALS

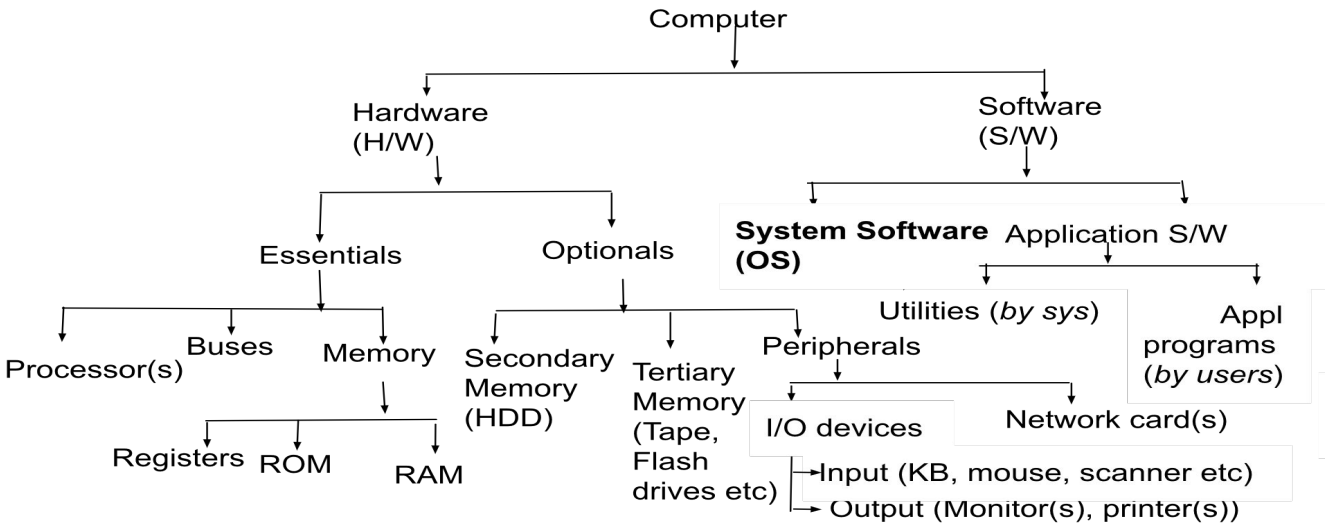


Fig 1.1: Components of computer system

What an Operating System does?

- The programs executed by computer hardware to perform a desired task are written in a **low-*level* machine language.** This language is often *difficult to understand*, and very *difficult to use.*
- High-level languages (C, C++, Java, Python etc.) cannot talk to the computer hardware directly.
- Operating systems come at this point to bridge the gap between the users (programmers or end-users) and the hardware.

USER VIEW:

- An OS provides an “easy-to-use” platform to the users over the “difficult-to-use” bare-bone hardware.
- For single user environment, the user monopolizes the resources (hardware resources like processors, memories, I/O devices etc or software) providing “ease of use” of the resources is the primary job of an OS
- For multi-user environment, resource utilization in an *equitable (*every user gets the chance to access all the resources according to her requirement*)* and *fair* (without any bias) manner is another important task of OS
-

SYSTEM VIEW:

- OS seen as a **Resource Allocator** or a **Control Program**

GENERATIONS OF OPERATING SYSTEMS

1. **First Generation (1940s - 50s):** It did not have any OS and was used to do simple calculations using plug-boards. The users had to manually feed the program and the data. Also called **Open Shop** phase.

2. **Second Generation (1955-1965):** The users were asked to prepare their 'job' (punched cards of programs and data) and submit to the operators who run the jobs on users' behalf. Also called **Closed Shop** or **batch processing** phase.

3. **Third Generation (1965-1980):** Multiple programs were simultaneously loaded in the main memory and interrupts enabled a processor to switch execution of one program to another. Paradigm shifted gradually to **multiprogramming,** then **time-sharing** to **concurrent programming.**

4. **Fourth Generation (1980-Now):** *Personal Computing* (PC) became a reality with focus shifting to single-user environment (user convenience preferred over resource utilization). GUI (*graphical user interface*), interaction through mouse clicks were included in the operating systems. PC was followed by **Distributed Systems**, then **Real-Time Systems** and then **Embedded Systems**.

GENERATIONS OF OPERATING SYSTEMS

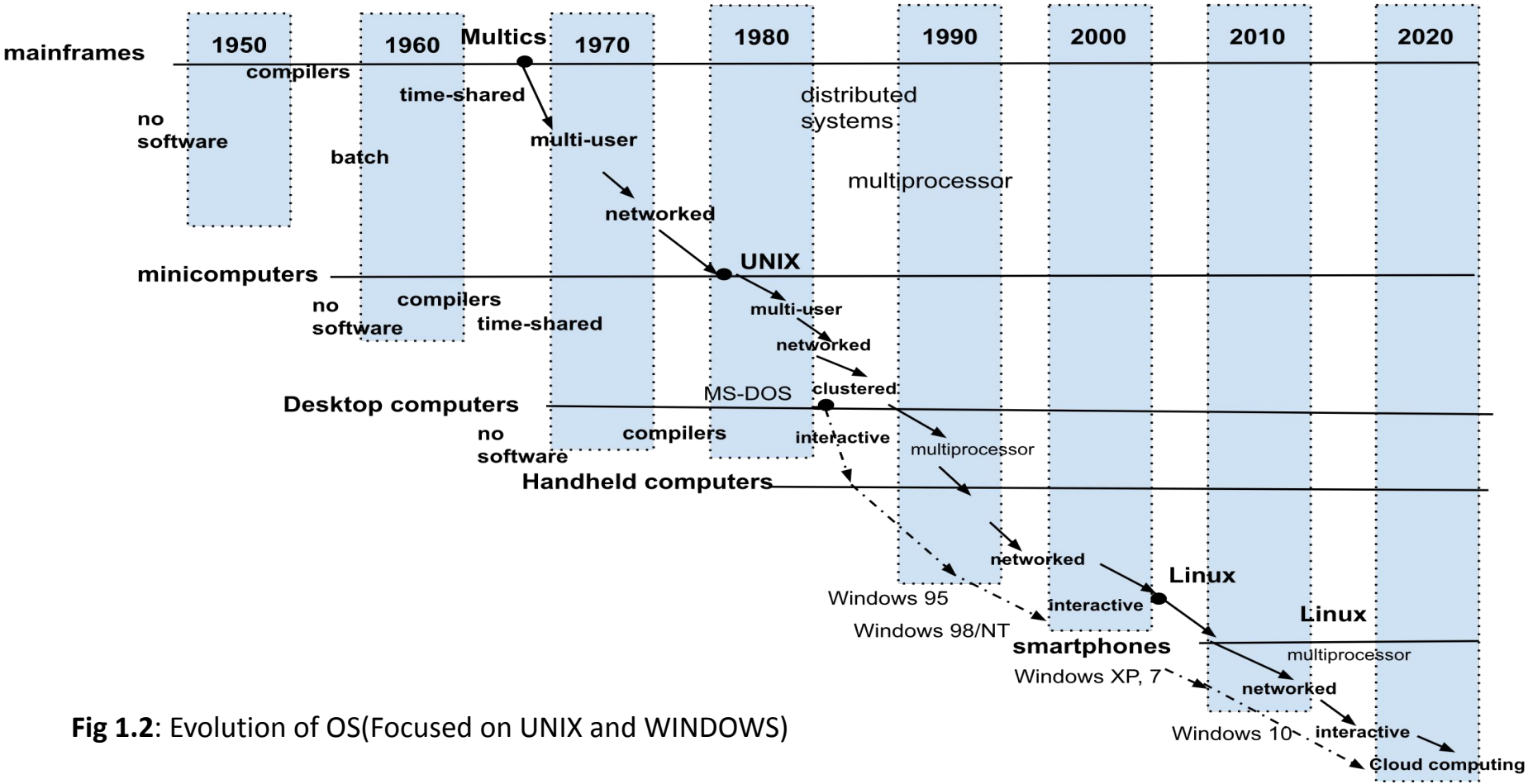


Fig 1.2: Evolution of OS(Focused on UNIX and WINDOWS)

TYPES OF OPERATING SYSTEMS

1. **Batch Systems**
2. **Interactive Systems**
3. **Hybrid Systems**
4. **Real-time Systems**
5. **Single Processor Systems**
6. **Multi-processor Systems**
7. **Multi-user Systems**
8. **Multiprogram Systems**
9. **Distributed Systems**
10. **Embedded Systems**

TYPES OF OPERATING SYSTEMS

1. **Batch Systems:**
   - Even today, the periodic tasks of the same types as payroll, billings, group membership mails are clubbed together, put in a queue for automatic execution.
   - The tasks need to be such that they do not require user intervention while being executed.
2. **Interactive Systems:**
   - These systems were designed to provide faster response time so that users can debug their programs.
   - Most of the prevalent OSs like different versions of Windows, UNIX and Linux are interactive ones.
3. **Hybrid Systems:**
   - Individual users can interactively execute their programs while the system accepts and runs batch programs when interactive load is low.
   - Operating systems of many large computers are hybrid ones.

TYPES OF OPERATING SYSTEMS

4. **Real-time Systems:**
   - They are of two basic types based on its response time: **hard real-time** and **soft-real time** systems.
   - A hard real-time OS has high consistency in completing a type of task (in the order of a few milliseconds).
   - A soft real time operating system allows relaxation (few hundred milliseconds with more variability or more ‘jitter’).

TYPES OF OPERATING SYSTEMS

5. Single Processor Systems:

- The systems having only one general purpose processor with a single core (core is the component that executes instructions and stores/fetches data in/from registers from the local storage) are known as single processor (or uniprocessor) systems.
- These systems, however, can have several special-purpose processors for managing specific I/O devices.
- These device-specific processors execute a limited number of instructions and are not used to run user programs.

6. Multi-processor Systems:

- One processor can house one or more CPUs in the processor chip while each CPU can have one or more cores.
- Each processor can have its own main memory or share the same along with peripheral devices.
- Multiprocessor systems are also called **tightly coupled systems**.
- Operating systems need to incorporate complex and sophisticated mechanisms for task scheduling, load balancing and synchronization among processors

TYPES OF OPERATING SYSTEMS

7. **Multi-user Systems:**
   - The operating system here must provide support to multiple users at the same time.
   - The OS allocates the resources in a fair and orderly manner to all the users. OS must ensure that each user works within her own authorized area (for her program and data) and does not transgress beyond her authority.
8. **Multiprogram Systems:**
   - When many application programs are allowed to run concurrently in a system the system is called a multiprogram system.
   - The main memory needs to accommodate all the application programs (either from a single user or multiple users) simultaneously.
   - Most of our computers are both multiuser and multiprogram systems. Most of today's single-user systems like mobile devices are also multiprogram ones.

TYPES OF OPERATING SYSTEMS

9. **Distributed Systems:**
   - Distributed systems are extensions of multiprocessor systems where several independent computers are connected through a network**.**
   - Users of a computer can use resources of another computer in the network.
   - Operating systems here help users to communicate with non-local computers through various mechanisms like message passing, remote procedure calls (RPC) and so on.
   - The user *feels* the networked system as a uniprocessor one.

10. **Embedded Systems:**
   - An embedded operating system is a small, special purpose OS embedded within a large machine to do a specific task.
   - Often Real-Time OSs (RTOS) are embedded within our car components, washing machines, and robots for executing certain tasks.
   - These OSs are small (a RTOS takes only a few MB space) and contain either no or extremely limited user interface.

- One OS can belong to several of the above types

OPERATING SYSTEM OPERATIONS

1. **Resource Management:**
   - Operating systems work as the **resource manager** of a computer. These resources are processes, memory, filesystem and I/O devices

   - **Process Management:**
     - A software program is a set of instructions that are executed in the CPU to accomplish a specific task. Programs are *passive* entities (as if a train rake). A program in execution is the *active entity* (as if a running train) and called a **process**.
     - Creation and deletion of processes
     - Scheduling of processes (or threads aka sub-processes) for CPU time
     - Suspending and resuming processes
     - Communication and coordination among several cooperating processes
     - Resolving contention among competing processes

OPERATING SYSTEM OPERATIONS

- **Memory Management:**
  - The main memory stores all the processes that are running in a system.
  - If we can accommodate many processes, the **degree of multiprogramming** increases, but a processor core can serve only one process at a time.
  - How many processes can be kept? for how long?
  - when a process needs to be pre-empted (removed) ?

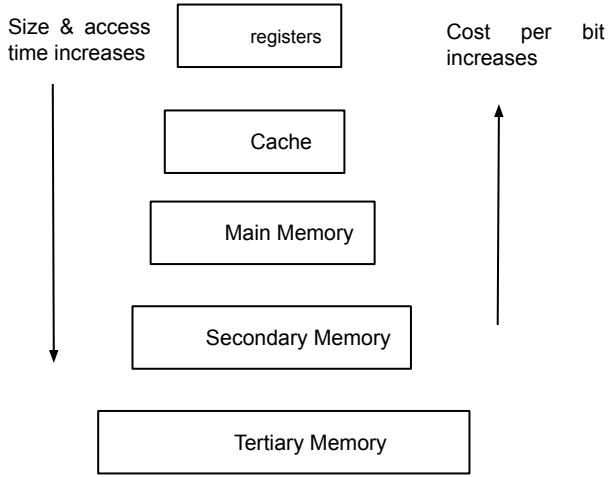


Fig 1.3: Memory Hierarchy

OPERATING SYSTEM OPERATIONS

- **File-system Management:**
  - Program code and data are stored temporarily in registers, cache and main memory during the program execution. But, in the long term, they are stored in secondary memory.
  - Formatting the media into file system type (e.g., DOS, windows, unix file system etc.)
  - Mapping files onto physical media
  - Creation, modification, and deletion of files
  - Creation, modification, organization, deletion of directories (and sub-directories)

  Copying (backing up) files and directories from one media to another.

- **I/O Management:**
  - Application programs use I/O devices (keyboard, mouse, touchpad, styler pen, monitor, scanner, I/O devices (keyboard, mouse, touchpad, styler pen, monitor, scanner, printer external disks etc.)
  - Different types of materials are used for different I/O devices and thus require different handling mechanisms
  - OS provides generic device drivers (softwares) to application programs and itself handles lower-level nitty-gritties through different device controllers (hardwares)
  - An I/O subsystem (part of OS) does the I/O device handling through buffer /cache management and spooling for data transfer and providing general and specific device drivers

OPERATING SYSTEM OPERATIONS

2. <u>Security and Protection:</u>

- Many users can concurrently use a computer in a multiuser, multiprogram environment. One user program can inadvertently or intentionally access a resource and/or execute a code that it is not supposed to.
- A computer system implements various kinds of protection schemes: some are at the hardware level and some at the software level.

- **Processing mode:**
  - At the hardware level, every processor supports at least two operation modes: **kernel** (or **supervisor, system** or **privileged**) **mode** and **user mode.**
  - In the kernel mode, a processor can execute all instructions like accessing all hardware and code from any user programs.
  - In user mode, the processor can execute instructions only from the designated memory regions and cannot access the hardware.
  - If the user program needs to access hardware or execute a code beyond its designated area it needs to raise a service request to the operating system through a **system call.**
  - A **System call** changes the processor mode from user to kernel.

OPERATING SYSTEM OPERATIONS

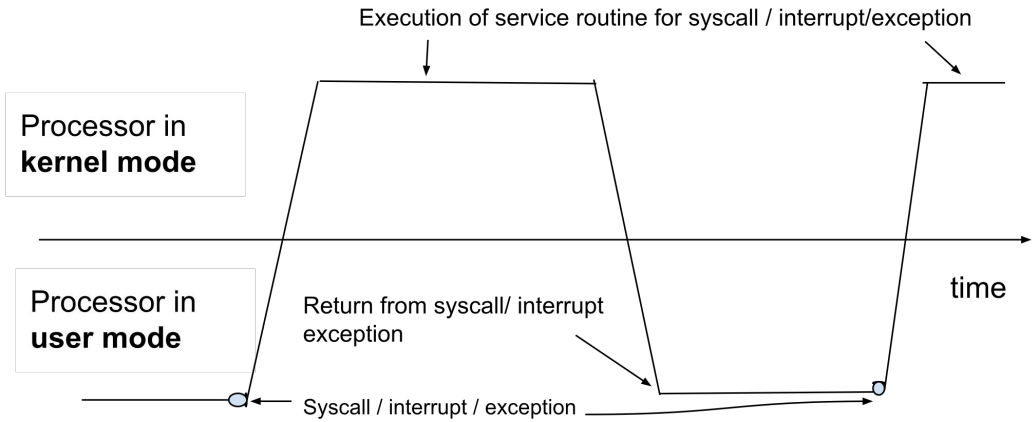


Fig 1.4: Dual mode Operation

OPERATING SYSTEM OPERATIONS

- **Address Space:**
  - Every process is allocated memory that may not be physically contiguous and not entirely loaded in the RAM while the program is being run.
  - virtual address space:The process sees conceptually (or virtually) the space as contiguous, such memory is called virtual memory.
  - Memory management unit or MMU translates the virtual address space to physical addresses so that the processor can access them.
- **Execution context**
  - Even though a system has adequate protection, it can fail and/or be vulnerable to inappropriate access to its resources.
  - Prevention of some of these attacks are the job of operating systems and some OSs offer some security measures for the same.
  - All modern OSs maintain a list of users and offer user-ids (UID). During login, UIDs are checked and only on successful authentication, users are allowed to use the operating system.
  - In some operating systems, users are grouped based on their privileges to access files and other resources.

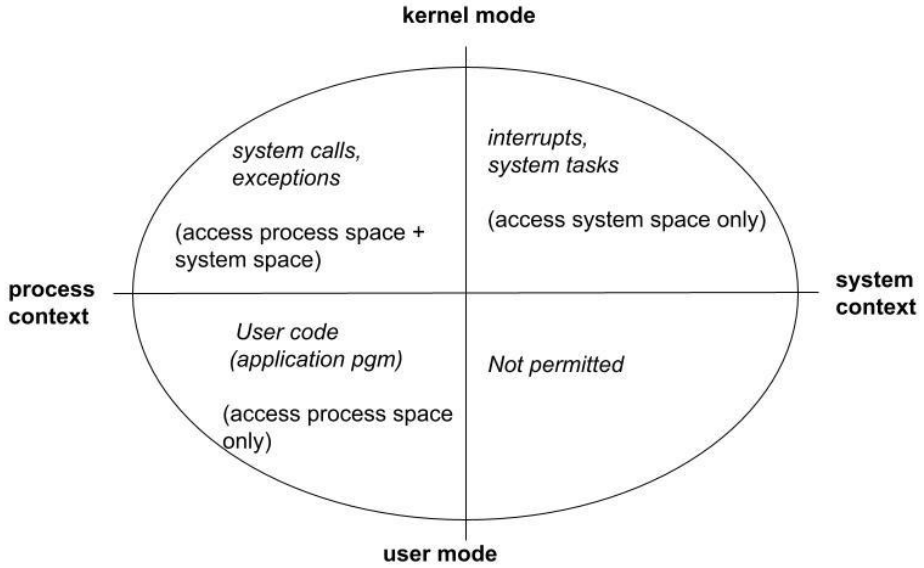


Fig 1.5: Relation among Process Mode

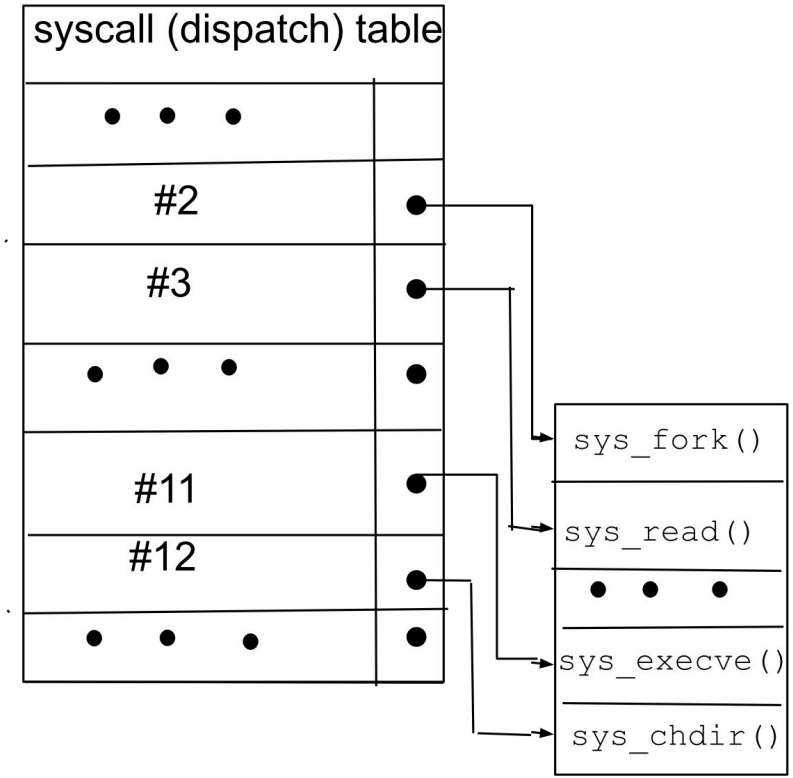
OPERATING SYSTEM SERVICES

- **Providing user interfaces**
- **Enabling program execution**
- **Enabling I/O handling**
- **Enabling file-system organization**
- **Enabling interprocess communication**
- **Detecting errors and enabling correction**
- **Allocation of resources**
- **Monitoring**
- **Protection and security**
- **SYSTEM CALLS (as well as exceptions and interrupts)**

SYSTEM CALLS

- User programs make system calls when they need to execute privileged instructions. These are like function calls offered by operating systems to user programs mainly for accessing the hardware.
- System calls are executed in *kernel mode* but initiated by the user process. Hence, they run in *process context* and access both *process space* and *kernel space (ref.* **Fig 1.5***).*
- A system goes into the kernel mode under three events:
  1. Interrupts
  2. Exceptions
  3. System calls
- Interrupts can come from any I/O device that may be active due to any process, not necessarily the currently running one.
- Exceptions are caused by illegal instructions that happen in the process space, and user mode and are synchronous events.
- System calls are very much like exceptions with the difference that they are lawful requests from the running process.

SYSTEM CALLS



Service type	Windows	Unix
Process Control	CreateProcess()	`fork()`
	ExitProcess()	`exit()`
	WaitForSingleObject()	`wait()`
	…	…
File Management	CreateFile()	`open()`
	ReadFile()	`read()`
	WriteFile()	`write()`
	CloseHandle()	`close()`
	…	…
Device Management	SetConsoleMode()	`ioctl()`
	ReadConsole()	`read()`
	WriteConsole()	`write()`
	…	…
Information Maintenance	GetCurrentProcessID()	`getpid()`
	SetTimer()	`alarm()`
	Sleep()	`sleep()`
	…	…
Communications	CreatePipe()	`pipe()`
	CreateFileMapping()	`shm_open()`
	MapViewOfFile()	`mmap()`
	…	…
Protection	SetFileSecurity()	`chmod()`
	SetSecurityDescriptorGroup()	`chown()`
	…	…

SYSTEM CALLS & APIs

- An application program may need several system calls to complete its intended task.
- For example, a simple program for copying some content from an input file to an output file involves several I/O operations or system calls :
  I. Opening the input file (1)
  II. Opening the output file (2)
  - Within a loop till there is content in (1)
    III. Reading the input file (1)
    IV. Writing on the output file (2)
  V. Closing the file (1)
  VI. Closing file (2)

- An OS provides a number of system call interfaces (SCIs) to the programming languages
- A programming language provides a number of application programming interfaces (APIs) through their libraries to application programs
- The APIs actually connect user programs to system calls through SCIs
- The SCIs are similar to railway ticket counters to avail different services of a railway station
- APIs are like third-party vendors for the same set of services

SYSTEM CALLS

User programs use APIs provided by library functions for making system calls.

Step 1: The compiler transfers it to the kernel's system call interface that changes the processor mode

Step 2: The interface raises an interrupt signal with the necessary number.

Step 3: In the system call table, it is resolved which system call is to be invoked.

Step 4: Once the appropriate ISR completes execution, control goes back to the system call interface.

Step 5: Return value is checked for error messages.

If no error is found, program state and other status variables are restored, with change in operating mode and control is returned to the user program so that execution can resume from the point where it was interrupted.

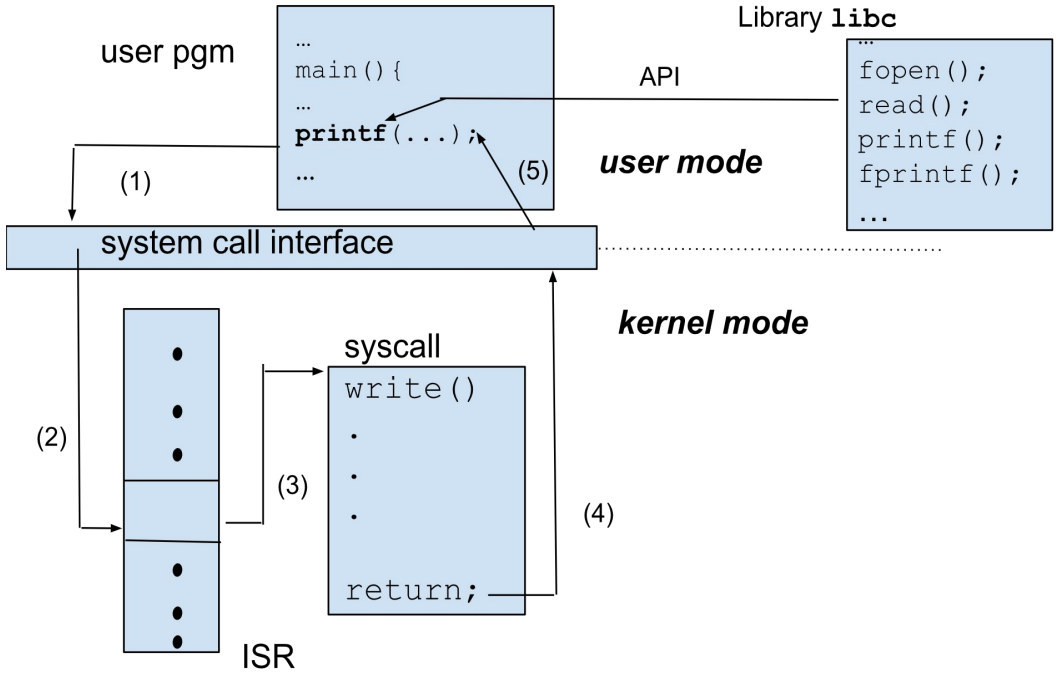


Fig 1.7: Interaction between API & System call

OPERATING SYSTEM STRUCTURE

- From a user's perspective, three key requirements from an os are:
  I. **Multiplexing of the resources**
  II. **Isolation of resources and processes**
  III. **Interaction among processes**
- Also from the designer's viewpoint, the operating system must have following two properties:
  I. **Portability**
  II. **Extensibility**

Monolithic Kernel

- The entire operating system runs as a single program and all the services that an OS offers to the applications are done in kernel mode.

•***Advantages:***

i. **Simplicity:** it is very simple in design.

ii. **Centralization:** a single code controls all the resources.

iii. **Close coupling:** all functionalities are in a single kernel space bypassing hassle of communications.

iv. **Performance:** it is fast and easy to maintain.

•***Disadvantages :***

i. **Size(fails in portability)**

ii. Adding a new functionality incurs the cost of compiling the entire code every time (**fails in extensibility)**

iii. For any bug in a particular service, the entire kernel may fail and abort all the running processes **(poor reliability).**

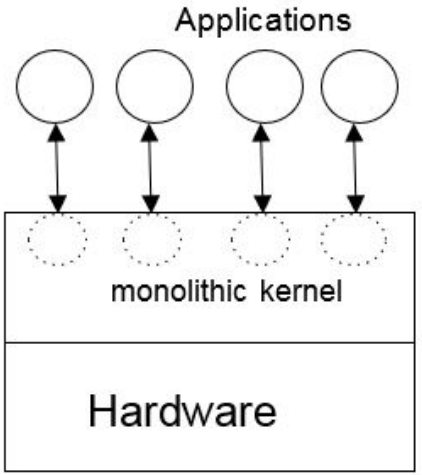


Fig 1.8: Monolithic Kernel

Microkernel

- Most of the OS services offered in user mode.
- Keep the amount of kernel mode code to a bare minimum.

Advantages:

i. **Good portability**
ii. **Greater reliability**
iii. **Easy extensibility**

Disadvantages:

i. **Increased communication**
ii. **Increased use of space:** every message is copied in two different process address spaces as well as in kernel space.
iii. **Poor performance**

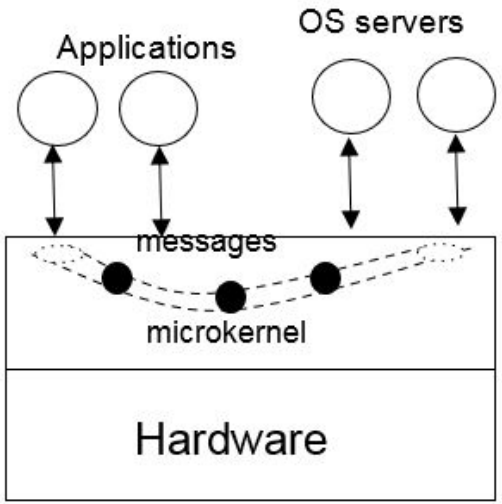


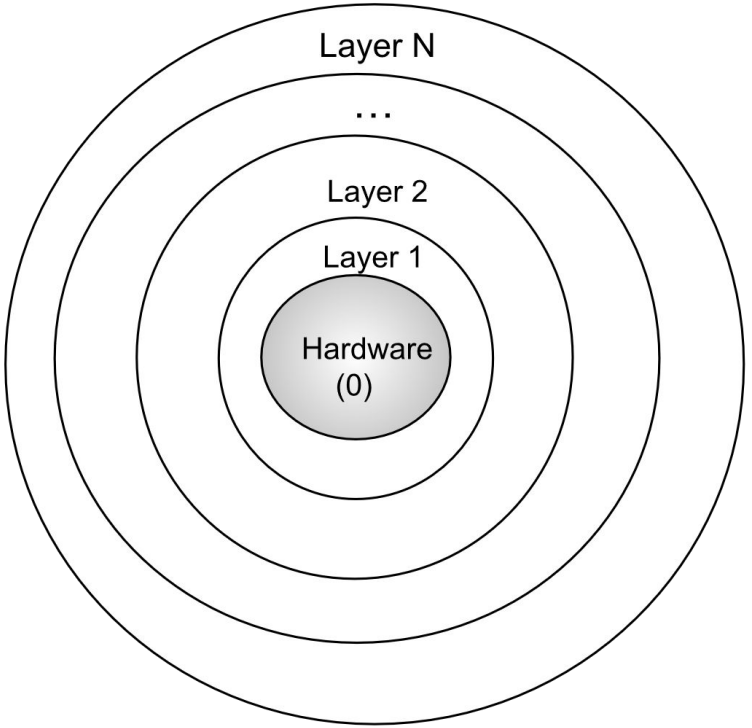
Fig 1.9: Microkernel

Layered Approach

- An alternative to both monolithic and microkernel approaches.
- The entire functionalities of an os are divided into well-defined layers.
- The innermost layer *(Layer 1)* deals with the bare-bone hardware *(Layer 0)* and the uppermost layer *(Layer n)* offers services to the applications.

- ***Advantages:***
  - **I. Portability**
  - **II. Isolation**
  - **III. Transparency**: upper layers need not know the details of lower layers.
- ***Disadvantages:***
  - **I. No consensus**: which functionalities should go at what layers.
  - **II. Difficult stratification**: not all functionalities can be elegantly broken down into the same number of layers.
  - **III. Increased communication**

Fig 1.10: Layered Architecture



OPERATING SYSTEM STRUCTURE

- **Modular Approach:**

- The approach is a mixture of microkernel and layered architecture.
- The kernel is divided into a set of core components and can link to several additional modules that can be dynamically loaded as and when required after bootup.
- These modules are called *loadable kernel modules* (LKMs).

- **Hybrid Approach:**

- Very few modern OS strictly implement one of the above approaches**.**
- There are combinations of two or more approaches.
- Example: **Linux** is monolithic broadly following the Unix philosophy. However, it has modular architectures. **Windows** is monolithic but contains some properties of microkernels

VIRTUAL MACHINES

- Virtual machines are non-real or illusory computing environments.
- The base has the hardware components, known as the **host.**
- The host is managed by a **virtual machine manager (VMM)** that virtualizes the computing environment.
- The OSs running on VMs are called **guest** OSs.

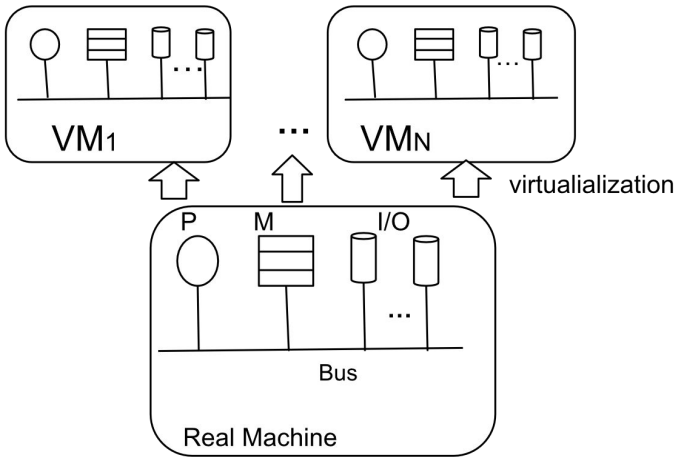


Fig 1.11: System level virtulization

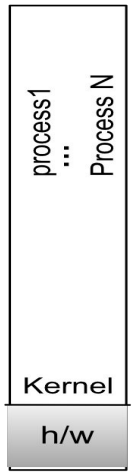


Fig 1.12: Single Os on Single machine

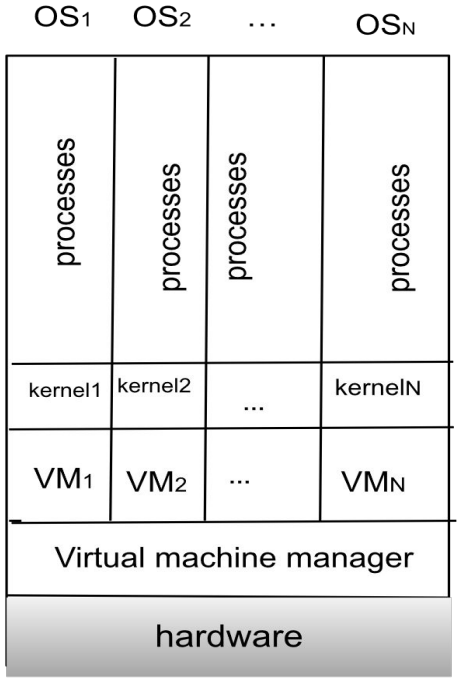


Fig 1.13: Virtualization on Single machine

Types of VMMs

1. **Type-0 Hypervisor:** Underlying hardware supports virtualization through creation and management of VMs.
2. **Type-1 Hypervisor:** VMMs interact with both host hardware and virtual machines as intermediaries
3. **Type-2 Hypervisor:** applications that run on some host OS and allow some other OS to run inside the application.

Virtual machines are very popular for cross-platform software development and testing due to

1. **Cost-effectiveness**
2. **Isolation**
3. **Consolidation**
4. **live migration**

<u>Other types of virtualization</u>

- **Para-virtualization:** Instead of virtualizing all hardwares, only a basic set of h/w are virtualized and OS is customized
- **Application level virtualization:** Java Virtual Machine (JVM) provides an execution environment independent of any OS. JVM runs on top of an OS.
- **Emulation:** Programs compiled in one instruction set architecture (guest) are converted into another instruction set architecture (host) to be run on host

OS CASE STUDIES

1. UNIX

- First developed in 1971, widely used in academia, research, and the business world.
- Multi-user, multiprogramming OS with simplicity in system and user design.
- Open-source since inception, enabling extensions and customizations.
- Kernel interacts directly with hardware and provides services to higher layers.
- Shells (e.g., bash, csh) offer command-line interface (CLI) and interact with kernel via system calls.
- Outer layers include compilers, editors, and user applications.

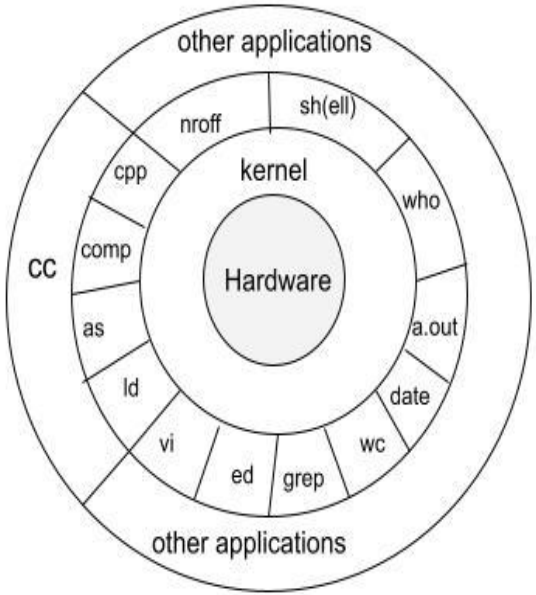


Fig 1.14: UNIX architecture

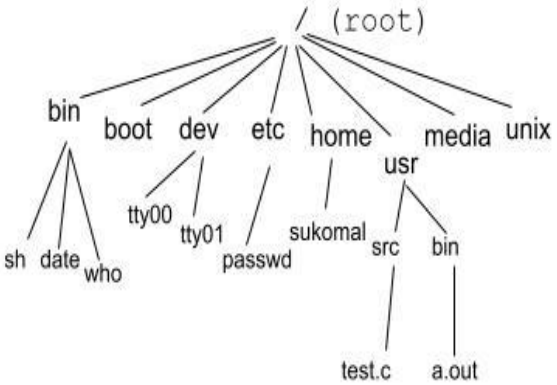


Fig 1.15: UNIX file system Organization

OS CASE STUDIES

1. WINDOWS:

- Developed by Microsoft, originating from MS-DOS in 1985.
- Dominant OS family with versions like Windows 10 and Windows Server 2016.
- Focuses on user-friendly GUIs and extensive application support.
- **User Mode:**
  - Runs user processes (applications, services), system processes, and environment subsystems.
  - Uses dynamic-link libraries (DLLs) for shared routines.
- **Kernel Mode:**
  - Includes core OS components like Executive (basic services), Kernel (low-level functions), and device drivers.
  - Provides GUI systems, HAL for hardware abstraction, and Hypervisor for virtualization.

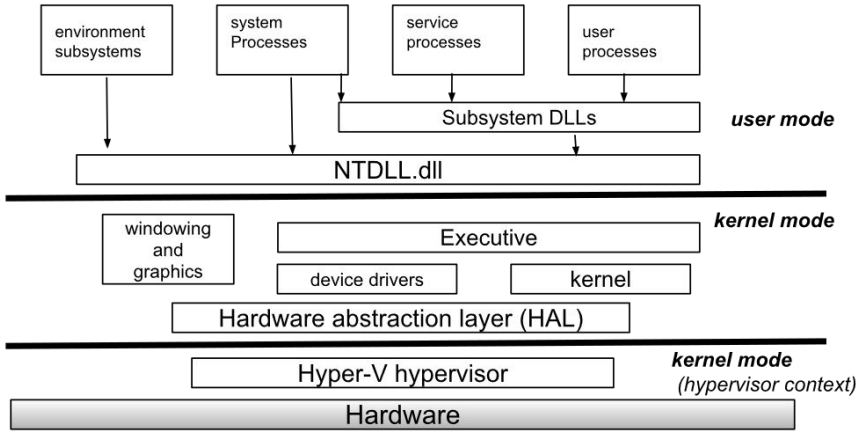


Fig 1.16: WINDOW architecture

Reference

[1] “OPERATING SYSTEMS”, Author: Dr. Sukomal Pal, Associate Professor Department of Computer Science & Engineering IIT (BHU), Varanasi, UP