# Introduction to Operating Systems

## The Objectives and Functions of Operating Systems

An **operating system (OS)** utilizes hardware resources of one or more processors to provide a wide range of services to system users. It also manages the secondary memory and the input/output (I/O) devices (Stallings, 2018). An operating system involves software and files that are installed on a computer so that it can boot and execute programs. The OS technically controls the hardware and coordinates its use among the various programs for various users. It includes the *kernel*, *administration tools*, and *system libraries* (Gregg, 2021).

The following can be considered as the objectives of an operating system (Stallings, 2018):

- An operating system makes a computer convenient to use.
- An operating system allows computer resources to be utilized in an efficient manner.
- Operating systems are generally constructed to effectively support the development, testing, and implementation of new system functions or software applications.

The most important collection of system programs comprises the operating system. It masks the details of the hardware from programmers while providing a convenient interface for using the system. The OS, as a computer interface, provides services to the following areas (Stallings, 2018):

- **Program development**: The OS provides a variety of facilities and services in the form of utility programs, such as editors and debuggers, for program development, which is also referred to as application program development tools.
- **Program execution**: The OS handles different processes, such as loading data into the memory and scheduling, which must be performed to execute programs.
- **Access to I/O devices**: The OS provides a uniform interface that hides complex sets of instructions for I/O devices' operation.
- **Control access to files**: The OS encompasses a detailed understanding of the structure of data contained in a file storage and can provide some protection mechanisms to control file access.
- **System resource access**: The OS controls access to the whole system and to some specific system resources through access functions and authorizations.
- **Error detection and response**: The OS can detect errors, such as device failure and software errors, and provide appropriate responses that would clear the error condition with the least impact on running applications.
- **Performance parameter monitoring**: The OS must be able to collect usage statistics from various processes, such as response time, that can be used as a basis for system performance improvements.
- **Instruction Set Architecture (ISA)**: The ISA defines the collection of machine language instructions that a computer can follow. The OS has access to additional machine language instructions that deal with managing a system's ISA.
- **Application Binary Interface (ABI)**: The ABI defines a standard for binary portability across programs. It defines the system call interface to the OS and the hardware resources and services available in a system through the ISA.
- **Application Program Interface (API)**: The API allows a program to the hardware resources and services available in a system through that ISA, supplemented with high-level language library calls. It also enables easier porting of application software.

The operating system is composed of instructions executed by the processor. During the instruction execution, the operating system decides how processor time is allocated and which computer resources are available for use. Thus, the OS, as a resource manager, can be described as a computer software that functions the same way as other programs in terms of its dependency on the processor (Stallings, 2018).

Operating systems evolve over time for the following reasons:

- Hardware upgrades or new types of hardware
- New or improved services
- Fixes to existing OS faults

## The Evolution of Operating Systems

**Serial Processing** (Stallings, 2018)

- From late 1940s to the mid-1950s, programmers interact directly with computer hardware since operating systems (OS) are not yet available.
- Computers are run from a console consisting of display lights, toggle switches, an input device, and a printer.
  - Programs in machine code were loaded via input device, such as a card reader.

- Error conditions are indicated by the display lights.
- If a program executes successfully, the output is printed.

- Serial processing presented two (2) main problems which are:
  1. **Scheduling**: Installations used a hardcopy sign-up sheet to reserve computer time. A user could sign up for an allocated block of time, but is unable to maximize the allocation. Thus, resulting in wasted computer processing time.
  2. **Set-up time**: A single program, called a *job*, involves numerous sequenced set-up processes that involve mounting and dismounting of tapes and setting up card decks. If an error occurred, the user had to go back to the beginning of the setup sequence. Thus, a considerable amount of time was spent in setting up the program before it even runs.

**Simple Batch Systems** (Silberschatz, Galvin & Gagne, 2018)

- To improve processor utilization, the concept of batch operating system was developed in the mid-1950s by *General Motors Research Laboratories*. It processed jobs in bulk, with predetermined input from files or other data resources.
- The main idea behind batch processing is the utilization of a piece of program known as the *monitor* or *resident monitor*. The monitor provides automatic job sequencing as indicated by the control cards. If a control card indicates that a program is to be run, the monitor loads the program into memory and transfers control to it. When the program completes, it transfers the control back to the monitor. Then, it reads the next control card, loads the appropriate program, and so on. This cycle is repeated until all control cards are interpreted for the job.

**Multi-programmed Batch Systems** (Stallings, 2018)

- This kind of operating system is more sophisticated compared to single-programmed systems. Several ready-to-run jobs must be kept in the main memory, requiring some form of memory management. If several jobs are ready to run, the processor decides which one to run, and that decision requires a specific algorithm for scheduling. It basically allows the processor to handle multiple jobs at a time.

**Time-sharing Systems** (Stallings, 2018)

- In a time-sharing system, multiple users simultaneously access the system through terminals, with the operating system interleaving the execution of each user program in a short burst time or quantum computation.
- If there are *n* number of users actively requesting service at exactly the same time, each user will only be allocated *1/n* of the effective compute capacity, excluding the operating system overhead.
- **Compatible Time-sharing System (CTSS)** was one of the first time-sharing operating systems that was developed. It was developed at the *Massachusetts Institute of Technology (MIT)* by the group *Project MAC*, for the *IBM 709* computer system.

Major theoretical advances in the development of operating systems mainly involve four (4) areas: *processes*, *memory management*, *information protection and security*, and *scheduling and resource management*. These areas extend to numerous key design and implementation issues of modern operating systems.

## Overview of Some Operating Systems (Stallings, 2018)

**Microsoft Windows Operating System**

- Microsoft initially used the name Windows for an operating environment extension to the primitive MS-DOS operating system (OS), which was a successful OS used on early personal computers.
- Windows separates the core OS software from the application-oriented software. The core OS software running in kernel-mode includes the Windows Executive, the Windows Kernel, device drivers, and the hardware abstraction layer. On the other hand, the remaining software running in user mode has limited access to system data. Windows is structured to provide support using a common set of kernel-mode components that lie beneath the OS environment subsystem.
- Windows has a highly modular architecture, wherein each system function is managed by only one component of the OS. By principle, any module can be removed, upgraded, or replaced without rewriting the entire system or its standard application program interfaces (APIs).

**Kernel Components of Microsoft Windows OS**

- **Windows Executive** – This contains the core OS modules/services for specific functions and provides an API for user-mode software. The following are its functions:
  - **I/O manager** - This provides a framework through which I/O devices are accessible to applications.
  - **Cache manager** – This improves the performance of file-based I/O by allowing recently referenced file data to reside in the main memory for quick access.

- **Object manager** – This creates, manages, and deletes Windows Executive objects that are used to represent resources such as processes, threads, and synchronization objects, while enforcing uniform rules for retaining, naming, and setting the security of the objects.
- **Plug-and-play manager** – This determines which drivers are required to support a particular device and loads those drivers.
- **Power manager** - This coordinates the power consumption among various devices. It can be configured to reduce power by shutting down idle devices or the entire system, or setting the processor to sleep.
- **Security reference monitor** – This enforces the access validation and audit-generation rules for all protected objects including files, processes, address spaces, and I/O devices.
- **Virtual memory manager** – This manages virtual addresses, physical memory, and paging files on a disk.
- **Process/thread manage**r – This creates, manages, and deletes processes and thread objects.
- **Configuration manager** – This is responsible for implementing and managing the system registry, which serves as the repository for both system-wide and per-user settings of various parameters.
- **Advanced local procedure call (ALPC) facility** – This implements an efficient cross-process call mechanism for communication between local processes implementing services and subsystems.

- **Windows Kernel** – This is considered the core software of the OS. It controls the execution of the processors. This manages thread scheduling, process switching, exception and interrupt handling, and multiprocessor synchronization. Note that the Kernel's own code does not run in threads.
- **Hardware abstraction layer (HAL)** – This isolates the OS from platform-specific hardware variances. It makes each computer's system bus, direct memory access (DMA) controller, interrupt controller, system timers, and memory controller look the same to the Windows Executive and Kernel components.
- **Device drivers** – These are dynamic libraries that extend the functionality of the Windows Executive. These include hardware device drivers that translate user I/O function calls into specific hardware device I/O request, and software components for implementing file systems, network protocols, and any other system extensions that need to run in kernel-mode.
- **Windowing and graphing system** – This implements the graphical user interface (GUI), such as the user interface controls and illustrations.

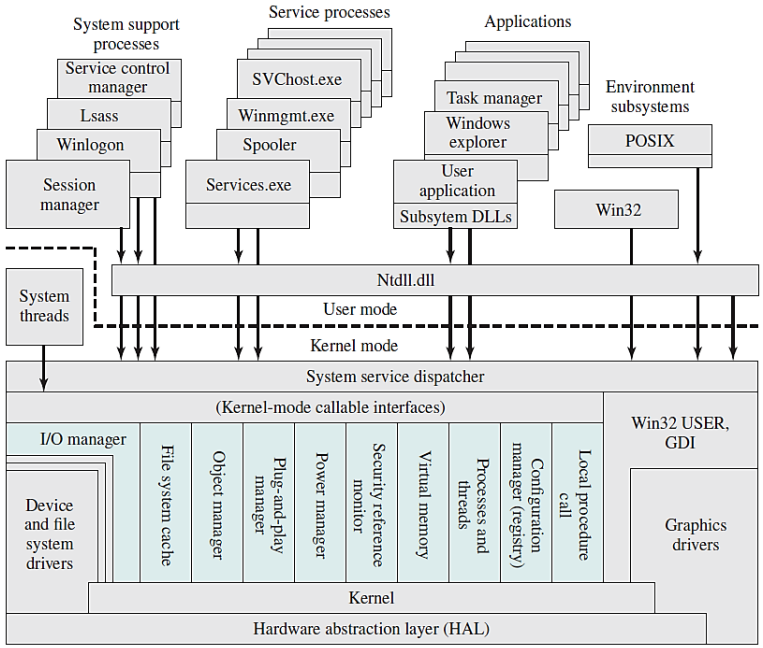


***Figure 1***. The internal architecture of Windows operating system.
*Source*: Operating Systems: Internal and Design Principles 9th ed., 2018 p. 102

**User-mode Processes**

1. **Special system processes**: These include user-mode services such as the session manager, the authentication subsystem, the service manager, and the log-on process.
2. **Service processes**: These involve services used by both Microsoft and other external software developers to extend system functionality and run background user-mode activity on a Windows system.
3. **Environment subsystems**: These provide different operating system environments and support Portable Operating System Interface (POSIX) and Win32 subsystems. Each environment subsystem contains processes shared among all applications.

4. **User applications**: These include executable (EXE) programs and dynamic link libraries (DLLs) that provide the functionality a user needs to make use of a system.

**Classic UNIX System**

- The classic UNIX system was developed at *Bell Labs* which was partly written in assembly language and was later converted into C language. It was initially called *Uniplexed Information and Computing Service (UNICS)*.
- The architecture of a classic UNIX has three (3) levels:
  - **User level** - focuses on the linked programs and libraries.
  - **Hardware level** - focuses on the physical hardware and interface.
  - **System kernel** - focuses on the overall operation of the system.

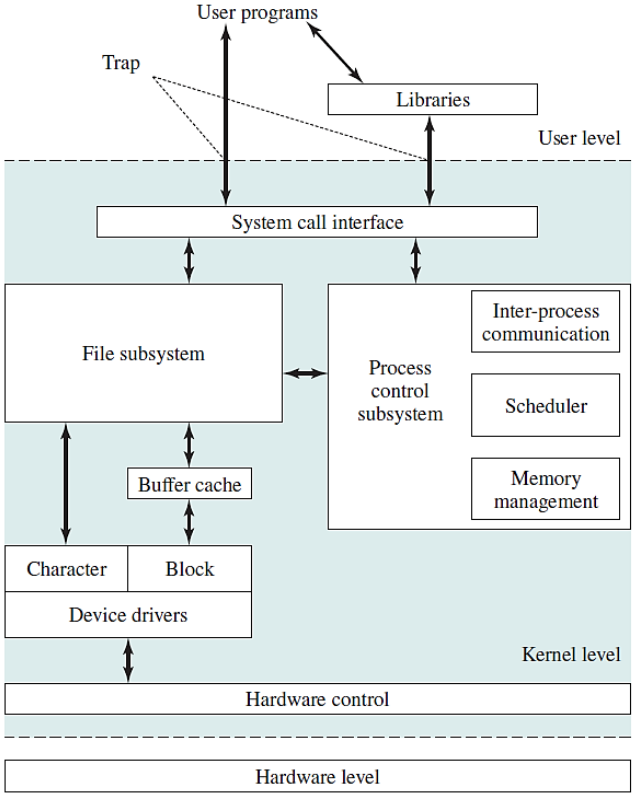


***Figure 2***. The internal architecture of a classic UNIX operating system.
*Source*: Operating Systems: Internal and Design Principles 9th ed., 2018 p. 110

- The classic UNIX is equipped with a number of user services and interfaces that are considered part of the system, which can be grouped into the shell. On the other hand, the OS still contains primitive routines that can directly interact with the hardware.
  - **Shell** – This supports system calls from applications, other interface software, and the components of the C compiler, such as the compiler itself, the assembler, and the loader.
- A classic UNIX is designed to run on a single processor and lacks the ability to protect its data structures from concurrent access by multiple processors. In addition, the system kernel of UNIX only contains few spaces for code reuse.

**Modern UNIX System**

- There was a need to produce a new system implementation that would unify significant innovations and modern design features, which lead to the development of the modular architecture of the modern UNIX system.
- The *Berkeley Software Distribution (BSD)* played an important role in the development of UNIX OS design theory and in releasing series of commercial UNIX products.

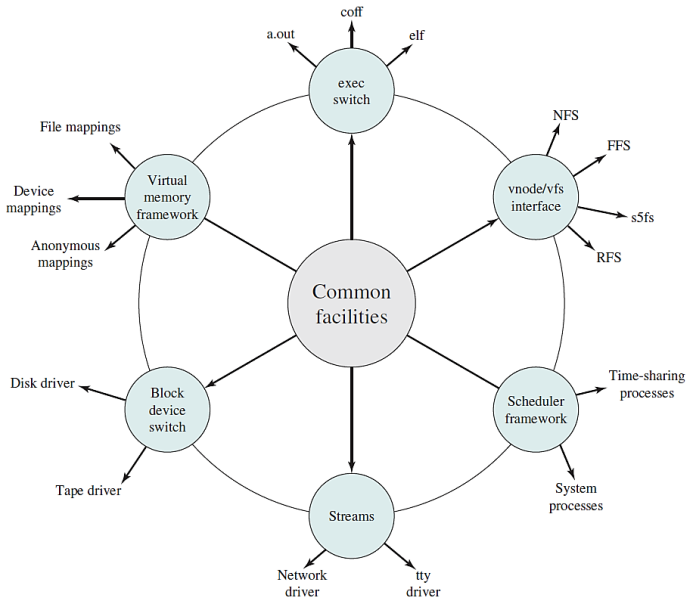


***Figure 3***. The internal architecture of a modern UNIX kernel.
*Source*: Operating Systems: Internal and Design Principles 9th ed., 2018 p. 111

- In a modern UNIX system, common facilities exist which serve as a core of the operating system, written in a modular fashion, that provide several functions and services needed by a number of OS processes.
  - **System V Release 4 (SVR4)** – A joint development by AT&T and Sun Microsystems. The system kernel of this operating system has a clean implementation. The new feature in this OS release includes real-time processing support, process scheduling classes, dynamically allocated data structures, virtual memory management, virtual file systems, and a preemptive system kernel. The SVR4 was developed to provide a uniform platform for commercial UNIX deployment, since it runs on processors ranging from 32-bit up to supercomputers.

**Linux Operating System**

- Linux started out as a UNIX variant for the IBM PC architecture, which was written by *Linus Torvalds*. Torvalds posted an early version of Linux on the Internet around 1991 and since then, numerous programmers and developers have collaborated over the Internet to improve the Linux operating system.
- Linux is a free open-source variant of the UNIX operating system, that is fully featured and runs virtually on different platforms. It is highly modular and easy to configure, which provides optimal performance in a wide variety of hardware platforms.

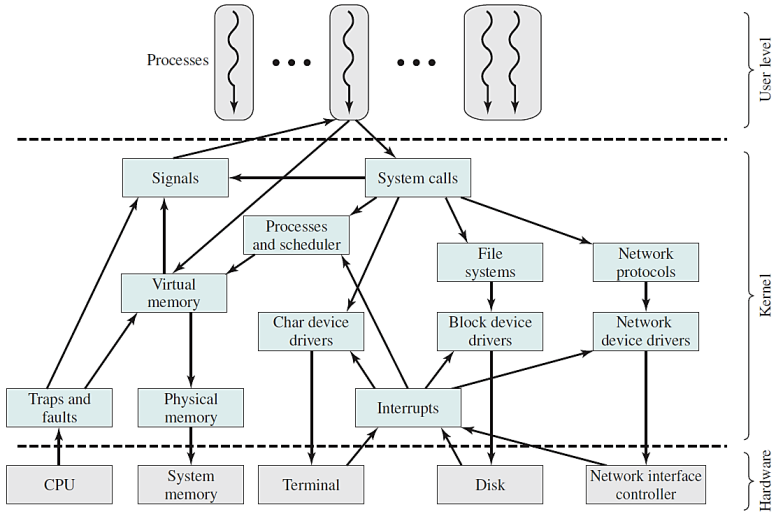


***Figure 4***. The structure of the Linux kernel components.
*Source*: Operating Systems: Internal and Design Principles 9th ed., 2018 p. 116

- The Linux operating system is composed of independent blocks called **loadable modules**, which can be automatically loaded and unloaded on demand. A loadable module is an object file whose code can be linked and unlinked from the kernel at runtime. It implements some specific functions, such as a file system and a device driver. In addition, a loadable module is executed in kernel mode on behalf of the current process. Below are the two (2) vital characteristics of the Linux loadable modules:
  1. **Dynamic linking** – The modules can be loaded and linked into the kernel while the kernel is already in memory and executing. A module can also be unlinked and removed from the memory at any time.
  2. **Stackable modules** – The modules are arranged in hierarchical order. Modules serve as libraries when they are referenced by other modules higher up in the hierarchy, and as clients when they reference modules further down.

**Kernel Components of Linux OS**

- Signals
- System calls
- Processes and scheduler
- Virtual memory
- File systems
- Network protocols
- Character device drivers
- Block device drivers
- Network device drivers
- Traps and faults
- Physical memory
- Interrupts

**Android Operating System**

- The Android operating system is a Linux-based system originally designed for mobile phones. It is defined as a software stack that includes a modified version of the Linux kernel, middleware, and some key applications.
- The kernel for an Android operating system does not contain drivers unsuitable for mobile environments, which technically makes it smaller. The Android relies on its Linux-based kernel only for the core system services, such as security, memory management, process management, network stack, and driver model.
- Android, Inc., which was bought by Google in 2005, initiated the Android OS development. Android has an active community of developers and enthusiasts who use the *Android Open-Source Project (AOSP)* source code to develop and distribute their own modified versions of the operating system.

**References:**
Gregg, B. (2021). *System performance: Enterprise and Cloud* (2nd ed.). Pearson Education, Inc.
Silberschatz, A., Galvin, P. & Gagne, G. (2018). *Operating systems concepts* (10th ed.). John Wiley & Sons, Inc.
Stallings, W. (2018). *Operating systems: Internal and design principles* (9th ed.). Pearson Education Limited