The first chapter reviews some of the basic concepts in computer system organization, introduces the general topics of operating systems and a few important concepts such as interrupt, multiprogramming, time sharing, virtualization, and cloud computing.

The main purposes or goals of an operating system:

- **User convenience**: it provides an environment for users or programmers to execute programs on computer hardware in a *convenient, safe, protected* and *efficient* manner.
- **Resource allocation**: it allocates computer resources, hopefully in a *fair* and *efficient* manner. The resources can be hardware such as CPU and main memory, or software such as signal and locks (to be discussed later). Thus, OS can be considered as a *control program*, which controls program execution, prevents users from misusing the resources, and handling I/O.

Storage Hierarchy

- A wide variety of storage devices in computer systems can be organized in a *hierarchy* according to speed, size and cost per unit or per byte. The higher levels in the hierarchy (i.e., closer to CPU) are more expensive with smaller size, but run much faster. As we move down the hierarchy, the cost per unit generally decreases, whereas the access time generally increases (slower) and capacity also increases.
- The main memory is a *volatile* storage device, which is the **only** large storage that the processor (CPU) can access directly. It holds both programs and data. In another word, the CPU accesses secondary storage indirectly.
- **Cache**, based on *temporal* and *spatial* locality, refers to a fast device that holds a subset of frequently used items from a slow device. Caching technique is widely used in computer systems and OS, whenever a subset of content needs to be stored in a faster device. For instance, part of a file directory - a kernel data structure for managing a file system stored on hard disks, can be cached in memory.
- The most common secondary-storage devices are hard-disk drives (HDDs) and nonvolatile memory (NVM) devices such as solid-state disks (SSDs).

A von Neumann architecture

- In this architecture, a typical instruction–execution cycle first fetches an instruction from memory (or cache) and stores the instruction in the *instruction register*. The instruction is then decoded and may cause operands (data) to be fetched from memory or cache and stored in data register(s). After the instruction on the operands is executed, the result may be stored back in memory or cache.
- One of the essential features in this architecture is that programs and data must be brought into memory before execution. Notice, however, that memory management unit or MMU (discussed later) sees only a stream of memory addresses. It does not need to know how they are generated, or what they are, whether instructions or data.

I/O Subsystems

- The basic hardware elements involved in I/O are buses, device controllers, and the devices themselves. The work of moving data between devices and main memory is performed by the CPU in programmed I/O or is offloaded to a DMA controller.
- The kernel module that controls a device is a **device driver**. The system call interface provided to applications is designed to handle several basic categories of hardware devices. The kernel I/O subsystem provides numerous services such as I/O scheduling, buffering, caching, spooling, device reservation, and error handling.

Interrupt

- All modern operating systems are **interrupt-driven**. An **interrupt** is a key way in which hardware interacts with an operating system. A hardware device can trigger an *interrupt* by sending a signal (through interrupt request lines) to the CPU indicating that some event requires CPU attention. For instance, an interrupt is used to signal the completion of an I/O operation.
- An **interrupt handler**, a specific OS program also referred to as *interrupt service routine*, is called to handle an interrupt when an interrupt occurs; this involves *context switch*.
- **Interrupt vector**, a *table* or an *array* that holds the addresses of interrupt handler for all interrupts. Each interrupt is assigned a unique ***interrupt number*** by the OS.
- A **trap** or **exception** is a software-generated interrupt. This can be used to call operating system routines or to catch arithmetic errors.
- It works roughly as follow (more details will be discussed later): The CPU hardware has wires called the ***interrupt-request lines*** that CPU senses or checks after executing every instruction. When the CPU detects a signal on an interrupt-request line, it reads the *interrupt number* and jumps to the *interrupt handler* by using that interrupt number as an index into the *interrupt vector*.
- Most CPUs have two interrupt request lines. One is the ***nonmaskable*** interrupt, which is reserved for events such as unrecoverable memory errors. The second interrupt line is ***maskable***: it can be turned off by the CPU before the execution of critical instruction sequences that must not be interrupted. The maskable interrupt is commonly used by device controllers to request service (which can delayed).
- The interrupt mechanism also implements a system of ***interrupt priority levels***. This enables the CPU to defer the handling of low-priority interrupts without masking all interrupts and makes it possible for a high-priority interrupt to preempt the execution of a low-priority interrupt.
- **DMA controller** is used to avoid programmed I/O, in which CPU moves one byte of data at a time between a (slow) device (such as keyboard and mouse) and memory. DMA is used for large chunk of data movement, e.g., between a disk drive and memory. It bypasses CPU to transfer data directly between I/O devices and the memory, which frees CPU for such data movement. DMA is standard in all modern computers, from smartphones to mainframe computers.

Multiprocessor Systems

- Multiprocessor systems are common in which each CPU can contain several computing cores. This increases the *throughput*, *reliability* and *economy of scale*.

- Symmetric multiprocessing or **SMP** treats all processors equally, and I/O can be handled by any processor. Asymmetric multiprocessing has one master CPU and the remainder CPUs are slaves. The master distributes tasks among the slaves, and I/O is usually done by the master processor only.
- Multiprocessor systems can save the overall cost without duplicating power supplies, housings, and peripherals. They can execute programs more quickly and can have increased reliability. But they are far more complex in both hardware design and software (i.e., parallel programming) than uni-processor or single-process systems.
- **Multicore systems**: multiple computing cores reside on a ***single chip***, which are more efficient than multiple processor chips, each with single core, because on-chip communication is faster than between-chip communication. In addition, one chip with multiple cores uses significantly less power than multiple single-core chips, an important issue for mobile devices as well as laptops (any devices running on battery).

Multiprogramming and Multitasking

- To best utilize the CPU, multiprogramming is used to allow several jobs to be brought into memory at the same time, and try to ensure CPU always has a job to execute.
- Multitasking or time-sharing is an extension of multiprogramming wherein CPU scheduling algorithms rapidly switch between processes, providing each user with a fast response time. Consequently, both require the support from OS.

Virtualization and Cloud Computing

- **Virtualization** is a technology that allows an OS to run as an application within another OS. It involves abstracting a computer's hardware into several different execution environments.
- The **virtual machine** or **VM** creates an *illusion* for multiple processes in that each process "thinks" that it runs on a dedicate CPU (processor) with its own memory.
- Cloud computing is a type of computing platform that delivers computing, storage, or/and application services on-demand over a network. Cloud computing often uses **virtualization** to provide its functionality. There are many different types of cloud environments, as well as a variety of services offered. Cloud computing may be either public, private, or a hybrid of the two. Additionally, cloud computing may offer applications, platforms, or system infrastructures as services.

Free and Open-source Operating System

- Free and open-source operating systems are available in source-code format.
- Free software is licensed to allow no-cost use, redistribution, and modification.
- GNU/Linux, FreeBSD, and Solaris are examples of popular open-source systems.