Chapter 2 summary

- Operating systems provide an environment for execution of programs and services to programs and users.
- One set of operating-system services provides functions that are helpful to the user:
    - **User interface** - Almost all operating systems have a user interface (UI).
    - **Program execution** - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally.
    - **I/O operations** - A running program may require I/O, which may involve a file or an I/O device.
    - **File-system manipulation** - The file system is of particular interest. Programs need to read and write files and directories, create, and delete them, search them, list file Information, permission management.
    - **Communications** – Processes may exchange information, on the same computer or on different computers over a network.
    - **Error detection** – OS needs to be constantly aware of possible errors.
    - **Resource allocation** - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them.
    - **Accounting** – To keep track of which users use how much and what kinds of computer resources.
    - **Protection and security** - The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other.
        - **Protection** involves ensuring that all access to system resources is controlled.
        - **Security** of system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts.
- CLI or **command interpreter** allows users to enter commands that performed by OS.
- **System calls** – programming interface to the services provided by the OS
    - Mostly accessed by programs via a high-level **Application Programming Interface (API)** rather than direct system call use.
- Typically, a number associated with each system call
    - **System-call interface** maintains a table indexed according to these numbers.
- The caller need know nothing about how the system call is implemented
    - Just needs to follow API and understand what OS will do as a result call.
    - Most details of OS interface are hidden from the programmer by API.
        - Managed by set of functions built into libraries included with compiler.
- Often, more information is required than simply identity of desired system call.
    - Exact type and amount of information vary according to OS and call.

- Three general methods used to pass parameters to the OS:
    - Simplest: pass the parameters in registers
        - In some cases, may be more parameters than registers.
    - Parameters stored in a block, or table, in memory, and block address passed as a parameter in a register.
        - This approach taken by Linux and Solaris.
    - Parameters **pushed**, onto the **stack** by the program and **popped** off the stack by the operating system.
- **Debugger** for determining **bugs, single step** execution.
- **Locks** for managing access to shared data between processes.
- System programs provide a convenient environment for program development and execution.
    - Some of them are simply user interfaces to system calls; others are considerably more complex
    - Most users' view of the operation system is defined by system programs, not the actual system calls
- Types of system programs:
    - File management
    - Status information
        - Some systems implement a **registry** - used to store and retrieve configuration information.
    - File modification
    - Programing-language support
    - Program loading and execution
    - Communications
    - Background services
    - Application programs
- User goals and System goals when designing and implementing OS
    - User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast.
    - System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient.
- Mechanisms determine how to do something; policies decide what will be done.
- Specifying and designing OS is highly creative task of **software engineering**.
- More high-level language easier to port to other hardware but slower.
- **Emulation** can allow an OS to run on non-native hardware.
- General-purpose OS is very **large program**.
- Simple structure (MS-DOS) – written to provide the most **functionality** in the least **space**
- UNIX – limited by hardware functionality, had limited structuring.

- The UNIX OS consists of two separable parts:
    - Systems programs
    - The kernel
- The operating system is divided into several layers (levels), each built on top of lower layers.
    - The bottom layer (layer 0) is the **hardware**; the highest (layer N) is the **user interface**.
    - With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers.
- Microkernel system structure – moves as much from the **kernel** into **user space**.
    - Communication takes place between user modules using **message passing**.
    - Benefits:
        - Easier to extend a microkernel.
        - Easier to port the operating system to new architectures.
        - More reliable (less code is running in kernel mode).
        - More secure.
    - Detriments:
        - Performance overhead of user space to kernel space communication.
- Most modern operating systems implement **loadable kernel modules**.
    - Overall, like layers but with more flexible.
- Most modern operating systems not one pure model.
    - Hybrid combines multiple approaches to address performance, security, usability needs.
- Apple mobile OS:
    - **Cocoa Touch** Objective-C API for developing apps
    - **Media services** layer for graphics, audio, video
    - **Core services** provides cloud computing, databases
    - **Core operating system** based on Mac OS X kernel
- Android:
    - Similar stack to iOS.
    - Based on Linux kernel but modified.
- **Debugging** is finding and fixing errors, or bugs
    - OSs generate **log files** containing error information.
- Failure of an application can generate **core dump** file capturing memory of the process.
- Operating system failure can generate **crash dump** file containing kernel memory.
- Beyond crashes, performance tuning can optimize system performance
    - Sometimes using **trace listings** of activities, recorded for analysis.
    - **Profiling** is periodic sampling instruction pointer to look for statistical trends.
- Performance tuning – improve performance by removing bottlenecks.

- **SYSGEN** program obtains information concerning the specific configuration of the hardware system.
- When power initialized on system, execution starts at a fixed memory location.
    - **Firmware** ROM used to hold initial boot code.
- Operating system must be made available to hardware so hardware can start it.
    - Small piece of code – **bootstrap loader**, stored in **ROM** or **EEPROM** locates the kernel, loads it into memory, and starts it.
    - Sometimes two-step process where **boot block** at fixed location loaded by ROM code, which loads bootstrap loader from disk.
- Common bootstrap loader, **GRUB**, allows selection of kernel from multiple disks, versions, kernel options.
- Kernel loads and system is then **running.**