***CHAPTER-1***

***INTRODUCTION***

------------------------------------------------------------------------------------------------------------------------------------------------------------------

Some operating systems are designed to be **convenient**, others to be **efficient**, and others to be some combination of the two.

Because an operating system is large and complex, it must be created piece by piece. Each of these pieces should be a well-delineated portion of the system, with carefully defined inputs, outputs, and functions.

------------------------------------------------------------------------------------------------------------------------------------------------------------------

**1.1 What Operating Systems Do**

* A computer system can be divided roughly into four components: the **hardware**, the **operating system**, the **application programs**, and the **users**.
* The hardware – the **central processing unit (CPU)**, the **memory** and the **Input/Output I/O devices** – provides the basic computing resources for the system.
* The application programs – such as word processors, spreadsheets, compilers, and web browsers – define the ways in which these resources are used to solve users’ computing problems.
* The operating system controls the hardware and coordinates its use among the various application programs for the various users. An operating system is similar to a government. Like a government, it performs no useful function by itself. It simply provides an environment within which other programs can do useful work.

**1.1.1 User View**

1. *Personal Computers (PCs)* – They are designed for a single user experience. The operating system is designed mostly for **ease of use**, with some attention paid to **performance** and none paid to resource utilization.
2. *Minicomputers* – They are designed for a multi-user experience. Many users access the same computer through multiple **terminals**. The operating system here is designed to maximize **resource utilization** – to ensure that all available CPU time, memory and I/O are used efficiently and fairly.
3. *Workstations and Servers* – These are also designed for a multi-user experience. Users have dedicated resources at their disposal but also share resources with other users. Therefore, their operating system is designed to compromise between individual usability and resource utilization.
4. *Mobile Computers* – The user interface generally features a **touch screen**, where the user interacts with the system by pressing and swiping fingers rather than using a physical keyboard and mouse.
5. *Embedded Computers* – They have little or no user view. Embedded computers in home devices and automobiles may have numeric keypads and may turn indicator lights on or off to show status, but their operating systems are designed primarily to run without user intervention.

**1.1.2 System View**

1. *Resource Allocator*: The operating system can be viewed as a **resource allocator**. It manages various CPU resources like CPU time, memory space, file storage space and I/O devices. It must efficiently and fairly allot resources to specific programs to avoid clashes.
2. *Control Program*: The operating system can also be viewed as a **control program**. A control program manages the execution of user programs to prevent errors and improper use of the computer.

**1.1.3 Defining Operating Systems**

**Moore’s Law**: Moore’s Law predicted that the number of transistors on an integrated circuit would double every eighteen months.

**Kernel**: The operating system is the one program running at all times on the computer, which is usually called the kernel.

**System Programs**: These are associated with the operating system but are not necessarily part of the kernel.

**Application Programs**: These include all programs not associated with the operation of the system.

**Middleware**: Mobile operating systems often include not only a core kernel but also middleware – a set of software frameworks that provide additional services to application developers.

------------------------------------------------------------------------------------------------------------------------------------------------------------------

**1.2 Computer-System Organization**

**1.2.1 Computer System Operation**

* A modern, general-purpose computer consists of one or more CPUs and a number of device controllers, connected through a common bus that provides access to shared memory.
* Each device controller is in charge of a specific type of device.
* **Memory Controller**: The CPU and device controllers can execute in parallel, competing for memory cycles. To ensure orderly access to the shared memory, a memory controller synchronizes access to the memory.
* **Bootstrap Program**: The initial program which is run when the computer is powered up or rebooted is known as the bootstrap program. It is stored in the **ROM** (Read-Only Memory) or **EEPROM** (Electrically Erasable Programmable Read-Only Memory), known by the general term **firmware**.
* *Responsibilities of the Bootstrap Program*:
  + *Initialization*: It initializes all aspects of the system, from CPU registers to device controllers to memory contents.
  + *OS*: It loads the operating system and starts executing the system. This is done by first locating the OS kernel and then loading it into the memory.
  + *Services*: Once the kernel is loaded and is executing, it provides services to the system and its users.
* **System Processes**: These are programs that are loaded into memory at boot time. These services are provided outside of the kernel.
* **System Daemons**: These processes run the entire time the kernel is running.
* **Interrupt**: An interrupt (either from the hardware or software) signals the occurrence of an event.
  + *Hardware*: Hardware may trigger an interrupt at any time, by sending a signal to the CPU, usually through the system bus.
  + *Software*: Software may trigger an interrupt by executing a special operation known as **system call** or **monitor call**.
* When the CPU is interrupted, it stops what it is doing and immediately transfers execution to a fixed location, which usually contains the starting address where the service routine for the interrupt is located. When the interrupt routine is completed, the CPU resumes the interrupted computation.
* **Pointers**: Since only a pre-defined number of interrupts is possible, a table of pointers (stored in low memory – first 100 locations or so) to interrupt routines can be used. The interrupt routine can then be indirectly called through the table.
* **Interrupt Vector**: The interrupt vector is an array storing locations of addresses the interrupt service routines.
* *Saving Address of the Interrupted Instruction*: This is essential, as after the interrupt is serviced, it should be loaded into the program counter, and the interrupted computation must resume as if the interrupt had not occurred. This address is stored either in a fixed location, or sometimes in the system stack.

**1.2.2 Storage Structure**

* **Random Access Memory**: General purpose computers run most of their programs from rewritable memory, called main memory or RAM.
* **Dynamic Random-Access Memory**: Main memory is commonly implemented in a semiconductor technology called DRAM.
* *Load Instruction*: It moves a byte or word from main memory to an internal register within the CPU.
* *Store instruction*: It moves the content of a register to main memory.

**Von-Neuman Architecture**

* A typical instruction-execution cycle, as executed on a system with VNA, first fetches an instruction from memory and stores that instruction in the **instruction register**.
* The instruction is then decoded and may cause operands to be fetched from memory and stored in some internal register.
* After the instruction on the operands has been executed, the result may be stored back in memory.
* Why programs and data cannot reside in the main memory permanently:

1. *Small Size*: Main memory is usually too small to store all needed programs and data permanently.
2. *Volatility*: Main memory is a volatile storage device that loses its contents when power is turned off or otherwise lost.

* **Secondary Storage**: It is an extension of the main memory. The main requirement of the secondary storage is that it be able to hold large quantities of data permanently.

*Storage-Device Hierarchy:*

* The wide variety of storage systems can be organized in a hierarchy according to speed and cost.
* The higher levels are expensive, but they are fast.
* As we move down the hierarchy, the cost per bit gradually decreases, whereas the access time generally increases.
* **Volatile storage**: Volatile storage loses its contents when the power to the device is removed.
* **Non-volatile storage**: Non-volatile storage does not lose its contents, even in the absence of battery and generator backup systems. Data is written into this for safekeeping.
* *Magnetic Disk*: It is the most common secondary-storage device. It provides storage for both programs and data. Most programs are stored on a disk until they are loaded into memory.
* *Solid-State Disks*: They are faster than magnetic disks and are non-volatile. Various types of solid-state disks are as follows:

1. *One type*: It stores data in a large DRAM array during normal operation but also contains a hidden magnetic hard disk and a battery for power backup.
2. *Flash Memory*: This is popular in cameras, PDAs (personal digital assistants), and robots. Flash memory is slower than DRAM but needs no power to retain its contents.
3. *NVRAM*: It is another form of non-volatile storage. It is DRAM with backup power. This memory can be as fast as DRAM and (as long as the battery lasts) is non-volatile.

**1.2.3 I/O Structure**

* A general purpose computer system consists of CPUs and multiple device controllers, that are connected through a common bus.
* Each device controller is in charge of a specific type of device.
* Depending on the controller, more than one device may be attached. For instance, seven or more devices can be attached to the **small computer-systems interface (SCSI)** controller.

Incomplete section, please complete it later

------------------------------------------------------------------------------------------------------------------------------------------------------------------

**1.3 Computer-System Architecture**

S

S

S

------------------------------------------------------------------------------------------------------------------------------------------------------------------