***CHAPTER-2***

***SYSTEM STRUCTURES***

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

**2.1 Operating System Services**

Functions that are helpful to the user:

1. *User interface*
   1. *Command-Line Interface*: It uses text commands and a method for entering them (like a keyboard).
   2. *Batch Interface*: Commands and directives to control commands are entered into files, and files are executed.
   3. *Graphical User Interface (GUI)*: The interface is a window system with a pointing device to direct I/O, choose from menus and make selections and a keyboard to enter text.
2. *Program Execution*: The system must load a program into its memory and run it.
3. *I/O Operations*: For efficiency and protection, the operating system must provide a means to do I/O rather than allowing users to directly control them.
4. *File-System Manipulation*: Programs need to read, write, delete, and search files and directories (by name). Operating systems must include permissions management to allow or deny access to file based on file ownership.
5. *Communication*: Communication between processes is implemented via shared memory or message passing.
6. *Error Detection*: The operating system needs to be detecting and correcting errors. For each type of error, the operating system must take the appropriate action to ensure correct and consistent computing.

Functions that ensure efficient operation of the system:

1. *Resource Allocation*: When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them. The operating system manages all these types of resources.
2. *Accounting*: A track of which users use how much and what kinds of computer resources must be tracked.
3. *Protection and Security*: Protection involves ensuring that all access to system resources is controlled, especially on a multiuser or networked computer system.

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

**2.2 User and Operating System Interface**

**2.2.1 Command Interpreters**

**Shells**: When there are multiple command interpreters to choose from, the interpreters are known as shells.

*Main Function*: It is to get and execute the next user-specified command.

**2.2.2 Graphical User Interface**

* *Desktop*: A mouse-based window-and-menu system is present and characterized by a desktop metaphor.
* *Icons*: The user moves the mouse to position its pointer on images, or icons, on the screen (the desktop). These represent programs, files, directories, and system functions.
* *Gestures*: Mobile systems, smartphones and handheld tablet computers typically use a touchscreen interface. Here, users interact by making gestures on the touchscreen – for example, pressing and swiping fingers across the screen.

**2.2.3 Choice of Interface**

1. *System Administrators and Power Users*: System administrators, who manage computers, and power users, who have deep knowledge of a system, use the command-line interface. Only a subset of system functions is available via the GUI, leaving the less common tasks to those who are command-line knowledgeable.

**Shell Scripts**: If a frequent task requires a set of command-line steps, those steps can be recorded into a file, and that file can be run just like a program. The program is not compiled into executable code but is rather interpreted by the command-line interface.

1. *Windows Users*: They are happy to use the Windows GUI environment.

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

**2.3 System Calls**

**System Calls**: System Calls provide an interface to the services made available by an operating system. They are written in C, C++, or sometimes using assembly-language instructions.

**Application Programming Interface (API)**: Application developers design programs according to API. The API specifies a set of functions that are available to an application programmer, including the parameters that are passed to each function and the return values the programmer can expect. Examples: Windows API for Windows systems, the POSIX API for POSIX-based systems, and the Java API for programs that run on the JVM.

**Library of Code**: A programmer can access an API via a library of code provided by the operating system. In case of UNIX and Linux or programs written in C, the library is called libc.

**System-Call Interface**: For programming languages, the run-time support system provides a system-call interface that serves as the link to system calls made available by the operating system. The system-call interface interprets function calls in the API and invokes the necessary system calls within the operating system.

Passing Parameters to the Operating System:

1. *Registers*: Parameters are passed in registers.
2. *Storage*: The parameters are stored in a block or table in memory and the address of the block is passed as a parameter in a register.
3. *Stack*: Parameters can also be pushed onto and popped off the stack by the operating system. This approach does not limit the number of parameters.

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

**2.4 Types of System Calls**

**2.4.1 Process Control**

1. End, Abort
2. Load, Execute
3. Create Processes, Terminate Processes
4. Get Process Attributes, Set Process Attributes
5. Wait for Time
6. Wait Event, Signal Event
7. Allocate and Free Memory

If a system call is made to terminate the currently running program abnormally, or if the program runs into a problem and causes an error trap, a dump of memory is sometimes taken and an error message generated.

The dump is written to disk and may be examined by a **debugger** – a system program designed to aid the programmer in finding and correcting errors or **bugs** – to determine the cause of the problem.

Two or more processes may share data. To ensure the integrity of the data being shared, operating systems provide system calls allowing a process to **lock** shared data. Then, no other process can acquire the data until the lock is released.

**2.4.2 File Management**

1. Create File, Delete File
2. Open, Close
3. Read, Write, Reposition
4. Get File Attributes, Set File Attributes

**2.4.3 Device Management**

1. Request Device, Release Device
2. Read, Write, Reposition
3. Get Device Attributes, Set Device Attributes
4. Logically Attach or Detach Devices

**2.4.4 Information Maintenance**

1. Get Time or Date, Set Time or Date
2. Get System Data, Set System Data
3. Get Process, File or Device Attributes
4. Set Process, File or Device Attributes

**Single Step**: Microprocessors provide a CPU mode known as single step, in which a trap is executed by the CPU after every instruction. The trap is usually caught by a debugger.

**2.4.5 Communication**

1. Create, Delete Communication Connection
2. Send, Receive Messages
3. Transfer Status Information
4. Attach or Detach Remote Devices

**Message-Passing Model**: In this model, the communication processes exchange messages with one another to transfer information.

*Hostname*: Each computer in a network has a hostname by which it is commonly known.

*IP-Address*: A host also has a network identifier, such as an IP-Address.

*Process Name*: Each process has a process name, and this name is translated into an identifier by which the operating system can refer to the process.

*Daemons*: Processes that receive connections are special purpose daemons, which are system programs provided for that purpose.

*Client*: The source of the communication is known as the client.

*Server*: The receiving daemon of the communication is known as server.

The client and the server exchange messages using system calls.

**Shared-Memory Model**: In this model, processes use system calls to create and gain access to regions of memory owned by other processes. Normally, the operating system tries to prevent one process from accessing another process’s memory. Shared memory requires that two or more processes agree to remove this restriction.

**2.4.6 Protection**

Protection provides a mechanism for controlling access to the resources provided by a computer system.

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

**2.5 System Programs**

**System Programs / System Utilities**: They provide a convenient environment for program development and execution.

1. File Management
2. Status Information: These programs ask the system for the date, time, amount of available memory or disk space, number of users, or similar status information. Some systems support a **registry**, which is used to store and retrieve configuration information.
3. File Modification
4. Programming-Language Support
5. Program Loading and Execution
6. Communication
7. Background Services: Constantly running system-program processes are known as **services**, **subsystems**, or **daemons**. One example is a service to listen for network connections in order to connect those requests to the correct process.

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

**2.6 Operating-System Design and Implementation**

**2.6.1 Design Goals**

At the highest level, the design of the system will be affected by the choice of hardware and the type of system: batch, time sharing, single user, multiuser, distributed, real time or general purpose.

Beyond this highest design level, the requirements can be divided into two basic groups: user goals and system goals.

1. *User Goals*: The system should be convenient to use, easy to learn and to use, reliable, safe and fast.
2. *Designer, Creator, Maintainer and Operator Goals*: The system should be easy to design, implement and maintain. It should be flexible, reliable, error free and efficient.

**Software Engineering**: General principles as to specifying and designing an operating system is discussed in software engineering. It is a highly creative task.

**2.6.2 Mechanisms and Policies**

**Mechanism**: Mechanisms determine *how* to do something.

**Policies**: Policies determine *what* will be done.

*Separation of Policy and Mechanism*: This is important for flexibility. Policies are likely to change across places or over time. A general mechanism, insensitive to changes in policy would be desirable. A change in policy would then require redefinition of only certain parameters of the system.

**2.6.3 Implementation**

Early operating systems were written in assembly language. Now most are written in higher-level languages such as C or C++. An operating system can be written in more than one language.

Advantages of using a higher-level language:

1. *General*: The code can be written faster, is more compact and easier to understand and debug.
2. *Compilation*: Improvements in compiler technology will improve the generated code for the entire operating system by simple recompilation.
3. *Portability*: The operating system becomes far easier to port – to move to some other hardware.

Disadvantage of using a higher-level language: Reduced speed and increased storage requirements are the only possible disadvantages.

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

**2.7 Operating-System Structure**

**Partition**: A system as large and complex as a modern operating system must be engineered carefully. A common approach is to partition the task into small components, or modules, rather than having one monolithic system.

**2.7.1 Simple Structure**

Many operating systems do not have well-defined structures. Frequently, such systems started as small, simple, and limited systems, and then grew beyond their original scope. MS-DOS is an example of such a system. It was written to provide the most functionality in the least space, so it was not carefully divided into modules.

UNIX OS: UNIX initially was limited by hardware functionality. It consists of two separatable parts – the kernel and system programs. The kernel provides the file system, CPU scheduling, memory management and other operating-system functions through system calls – this is an enormous amount of functionality to be combined into one level. This monolithic structure was difficult to implement and maintain. It had a distinct performance advantage however: there is very little overhead in the system call interference or in communication within the kernel.

**2.7.2 Layered Approach**

The operating system is broken into a number of layers or levels. The bottom layer (layer 0) is the hardware, and the highest level (layer N) is the user interface.

A typical operating-system layer (say layer M) consists of data structures and a set of routines that can be invoked by higher-level layers. Layer M, in turn, can invoke operations on lower-level layers.

*Advantage of Layered Approach*: The main advantage is the simplicity of construction and debugging. The layers are selected so that each uses functions and services of only lower-level layers. This approach simplifies debugging and system verification.

*Data Abstraction*: Each layer is implemented only with operations provided by lower-level layers. A layer does not need to know how these operations are implemented, it only needs to know what these operations do.

*Difficulty with Layered Approach*: The major difficulty involves defining the various layers. Careful planning is necessary. For example, the device driver for the backing store (disk space used by virtual-memory algorithms) must be at a lower level than the memory-management routines, because memory management requires the ability to use the backing store.

*Less Efficiency*: They tend to be less efficient than other types. For instance, when a user program executes an I/O operation, it executes a system call that is trapped to the I/O layer, which calls the memory management layer, which in turn calls the CPU-scheduling later, which is then passed to the hardware. At each layer, the parameters may be modified, data may need to be passed, and so on. Each layer adds overheat to the system call. The net result is a system call that takes longer than does one on a nonlayered system.

**2.7.3 Microkernels**

An operating system called Mach was developed, that modularized the kernel using the microkernel approach. This method structures the operating system by removing all non-essential components from the kernel and implementing them on a system and user-level programs.

*Main Function*: The main function of the microkernel is to provide communication between the client program and the various services that are also running in user space. Communication is provided through message passing.

*Advantage*: It makes extending the operating system easier. All services are added to user space and consequently do not require modification of the kernel. When the kernel does have to be modified, the changes tend to be fewer, because the microkernel is a smaller kernel. The resulting operating system is easier to port from one hardware to another. The microkernel also provides more security and reliability.

*Disadvantage*: The performance of microkernels can suffer due to system-function overhead.

**2.7.4 Modules**

**Loadable Kernel Modules**: Here, the kernel has a set of core components and links in additional services via modules, either at boot time or during run time. The idea of the design is for the kernel to provide core services while other services are implemented dynamically, as the kernel is running.

*Resemblance with Layered Approach*: Each kernel section has defined, protected interfaces, but it is more flexible than a layered system because any module can call any other module.

*Resemblance with Microkernel Approach*: The primary module has only core functions and knowledge of how to load and communicate with other modules, but it is more efficient, because modules do not need to invoke message passing in order to communicate.

**2.7.5 Hybrid Systems**

Operating systems combine different structures, resulting in hybrid systems that address performance, security, and usability issues.

Linux and Solaris are both monolithic, because having the operating system in a single address space provides very efficient performance. However, they are also modular, so that new functionality can be dynamically added to the Kernel.

Windows is largely monolithic as well, but it retains some typical behaviour of microkernel systems, including providing support for separate subsystems (known as operating system **personalities**) that run as user-mode process. Windows systems also provide support for dynamically loadable kernel modules.

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

**2.8 Operating-System Debugging**

**Debugging**: Debugging is the activity of finding and fixing errors in a system, both in hardware and in software. Performance problems are considered bugs, so debugging can also include **performance tuning**, which seeks to improve performance by removing processing **bottlenecks**.

**2.8.1 Failure Analysis**

**Log File**: If a process fails, most operating systems write the error information to a log file to alert system operators or users that the problem occurred.

**Core Dump**: The operating system can take a capture of the memory of the process and store it in a file for later analysis – this is known as code dump.

**Crash**: A failure in the kernel is called a crash. When a crash occurs, error information is saved to a log file, and the memory state is saved to a crash dump.

**2.8.2 Performance Tuning**

Performance tuning seeks to improve performance by removing processing bottlenecks. To identify bottlenecks, we must be able to monitor system performance. Thus, the operating system must have some means of computing and displaying measures of system behaviour. In a number of systems, the operating system does this by producing **trace listings** of system behaviour and running an analysis program later.

Another approach to performance tuning uses single-purpose, interactive tools that allow users and administrators to question the state of various system components to look for bottlenecks.

**2.8.3 D-Trace**

D-Trace is a facility that dynamically adds probes to a running system, both in user processes and in the kernel. These probes can be queried via the D programming language.

D-Trace is composed of a compiler, a framework, **providers** of **probes** within that framework, and **consumers** of those probes. The probes are stored in hash-table data structure that is hashed by name and indexed according to unique probe identifiers.

**Enabling control blocks (ECBs)**: Within the kernel, actions called ECBs are performed when probes fire.

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

**2.9 Operating System Generation**

**System Generation (SYSGEN)**: Operating systems are designed to run on any of a class of machines at variety of sites with a variety of peripheral configurations. The system must then be configured or generated for each specific computer site; a process sometimes known as system generation.

The following information must be determined:

1. What CPU is to be used? What options are installed?
2. How will the boot disk be formatted?
3. How much memory is available?
4. What devices are available?
5. What operating-system options are desired?

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

**2.10 System Boot**

**Booting**: The procedure of starting a computer by loading the kernel is known as booting the system.

**Bootstrap Program / Bootstrap Loader**: It is a small piece of code that locates the kernel, loads it into the main memory, and starts its execution. The bootstrap program can perform a variety of tasks:

1. *Diagnostics*: One task is to run diagnostics to determine the state of the machine. If the diagnostics pass, the program can continue with the booting steps.
2. *Initialisation*: It can also initialize all aspects of the system, from CPU registers to device controllers and the contents of the main memory.

**Read Only Memory (ROM)**: The initial bootstrap program is generally in the form of ROM, because the RAM is in an unknown state at system startup. ROM is convenient because it needs no initialization and cannot easily be infected by a computer virus.

**Erasable Programmable Read Only Memory (EPROM)**: The problem with ROM is that changing the bootstrap code requires changing the ROM hardware chips. This problem is resolved by using EPROM, which is read-only except when explicitly given a command to become writable.

**Firmware**: All forms of ROM are also known as firmware, since their characteristics fall somewhere between those of hardware and those of software.

**Boot block**: For large operating systems, or for systems that change frequently, the bootstrap loader is stored in firmware, and the operating system is on disk. In this case, the bootstrap runs diagnostics and has bit of code that can read a single block at a fixed location (say block zero) from disk into memory and execute the code from that boot block.

**Boot Disk / System Disk**: All of the disk-bound bootstrap, and the operating system itself, can be easily changed by writing new versions to disk. A disk that has a boot partition is called a boot disk / system disk.

**Running State**: When the full bootstrap program has been loaded, it can traverse the file system to find the operating system kernel, load it into memory, and start its execution. It is only at this point when the system is said to be running.

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