Chapter-2(read slide topics)

**Operating-System Services**

1. \*\*User Interface:\*\* Operating systems provide different types of user interfaces, including Command-Line Interfaces (CLI), Batch Interfaces, and Graphical User Interfaces (GUI). These interfaces allow users to interact with the system, issue commands, and perform tasks.

2. \*\*Program Execution:\*\* The operating system is responsible for loading programs into memory and executing them. It manages the execution of programs and handles their termination, whether normal or due to errors.

3. \*\*I/O Operations:\*\* Operating systems mediate input and output operations between running programs and I/O devices. This includes managing files and devices like disks, keyboards, and displays. Users typically cannot control these devices directly for efficiency and security reasons.

4. \*\*File-System Manipulation:\*\* Users and programs need to read, write, create, delete, and manage files and directories. Operating systems provide file systems that offer these functionalities. They also handle file permissions to control access.

5. \*\*Communications:\*\* Operating systems enable communication between processes, whether on the same computer or across a network. This includes shared memory and message passing mechanisms to exchange data between processes.

6. \*\*Error Detection:\*\* The operating system constantly monitors for errors, whether they occur in hardware (e.g., memory errors), I/O devices (e.g., network failures), or user programs (e.g., illegal memory access). It takes appropriate actions to maintain system integrity, such as terminating a faulty process or halting the system if necessary.

7. \*\*Resource Allocation:\*\* When multiple users or tasks run concurrently, the operating system manages resource allocation. This includes CPU scheduling, memory management, and allocation of I/O devices to ensure fair and efficient resource utilization.

8. \*\*Accounting:\*\* Operating systems keep track of resource usage by different users and processes. This information can be used for billing users, monitoring system performance, and making informed decisions about resource allocation.

9. \*\*Protection and Security:\*\* Protecting data and ensuring system security are critical functions. The operating system controls access to resources, enforces user authentication (e.g., password protection), and guards against unauthorized access, including external threats like network attacks.

In summary, operating systems play a crucial role in providing a seamless and secure computing environment. They offer services that simplify interactions for users and efficiently manage system resources for optimal performance while safeguarding against errors and unauthorized access. These services vary depending on the type and purpose of the operating system but are fundamental to its operation.

**User and Operating-System Interface**

See book

**System Calls**

Certainly! This passage explains the concept of system calls and how they are used by programs to interact with the operating system. Let's break down the key points:

1. \*\*What Are System Calls:\*\* System calls are a way for programs to request services and functionality provided by the operating system. These services can include actions like reading from files, writing to devices, creating processes, and managing hardware resources.

2. \*\*Programming Interface for System Calls:\*\* System calls are generally accessed through routines written in programming languages like C and C++. These routines are part of libraries provided by the operating system. Programmers can invoke these routines to make system calls without having to interact directly with low-level hardware.

3. \*\*Example: Reading and Copying Files:\*\* To illustrate the use of system calls, the passage describes a simple program that reads data from one file and copies it to another file. This operation involves multiple system calls, including opening files, reading data, writing data, handling errors, and closing files.

4. \*\*Parameters for System Calls:\*\* System calls often require specific parameters to provide information about the desired action. These parameters can include things like file names, buffer addresses, and data sizes. Depending on the operating system, these parameters can be passed in different ways:

- \*\*Registers:\*\* Some parameters can be passed directly in registers, which are small, fast storage locations within the CPU.

- \*\*Memory Blocks or Tables:\*\* When there are more parameters than available registers, the parameters are stored in a block or table in memory. The address of this block is then passed as a parameter.

- \*\*Stack:\*\* Parameters can also be pushed onto the program's stack and popped off by the operating system.

5. \*\*API (Application Programming Interface):\*\* Most programming languages provide an API to access system calls. The API defines the functions that programmers can use, including their parameters and return values. Programmers interact with the API rather than making direct system calls. The API functions are implemented in libraries that manage the details of the system call invocation and communication with the operating system.

6. \*\*Portability:\*\* Programming using an API offers portability advantages. Programs written to conform to an API can generally run on any system that supports that API, regardless of the specific underlying operating system. This is because the API provides a standardized way to make system calls, abstracting the differences between operating systems.

7. \*\*Behind the Scenes:\*\* The system-call interface intercepts calls made by programs to the API functions and invokes the corresponding system calls within the operating system's kernel. The system-call interface maintains a table of system calls indexed by numbers, and it translates API calls into the appropriate kernel-level operations.

In summary, system calls are a fundamental mechanism for programs to interact with the operating system. They enable programs to access various operating system services and functionalities. Programmers typically work with an API provided by the operating system, which abstracts the details of system call invocation and makes their programs more portable across different systems.

**Types of System Calls**

1. \*\*Process Control (Process-Related System Calls):\*\*

- `end()` and `abort()`: These system calls allow a program to terminate either normally or abnormally.

- `load()` and `execute()`: These system calls enable a program to load and execute another program.

- `create process()` and `terminate process()`: These system calls create and terminate processes, respectively.

- `get process attributes()` and `set process attributes()`: These system calls retrieve and modify process attributes.

- `wait for time()`: This system call allows a program to wait for a specified amount of time.

- `wait event()` and `signal event()`: These calls are used for synchronization and communication between processes.

- Memory-related calls: There are also system calls for memory allocation and deallocation.

2. \*\*File Management (File-Related System Calls):\*\*

- `create file()` and `delete file()`: These system calls create and delete files.

- `open()`, `close()`, `read()`, `write()`, `reposition()`: These system calls are used for file manipulation.

- `get file attributes()` and `set file attributes()`: These calls retrieve and modify file attributes.

3. \*\*Device Management (Device-Related System Calls):\*\*

- `request device()` and `release device()`: These calls are used to request and release access to devices.

- `read()`, `write()`, `reposition()`: These are device manipulation calls similar to file calls.

- `get device attributes()` and `set device attributes()`: These calls retrieve and modify device attributes.

4. \*\*Information Maintenance (Information-Related System Calls):\*\*

- `get time` or `date()` and `set time` or `date()`: These calls provide access to system time and date.

- `get system data()` and `set system data()`: These calls retrieve and modify system-level information.

- `get process`, `file`, or `device attributes()` and `set process`, `file`, or `device attributes()`: These calls deal with attributes related to processes, files, and devices.

5. \*\*Communication (Communication-Related System Calls):\*\*

- These system calls facilitate inter-process communication (IPC) and can involve creating, deleting, sending, receiving, or managing communication connections.

- The communication models include message passing and shared memory.

6. \*\*Protection (Protection-Related System Calls):\*\*

- Protection system calls control access to resources and specify permissions.

- `set permission()` and `get permission()`: These calls manage permissions for resources.

- `allow user()` and `deny user()`: These calls control user access to resources.

These system calls provide a wide range of functionality for programs and processes to interact with the operating system and with each other. Depending on the operating system, the specific names and details of these system calls may vary, but the general concepts remain consistent across different platforms.

**System Programs**

See book

**System Boot**

1. Power-On and Reset:

- When you power on or reboot your computer, the CPU is initially in an unknown state.

- The CPU's instruction register is loaded with a predefined memory location, typically pointing to a special area in read-only memory (ROM).

2. Bootstrap Program (Bootloader):

- At the predefined memory location in ROM, the bootstrap program (or bootloader) resides.

- The bootstrap program's purpose is to initiate the boot process.

- It performs tasks like running diagnostics to check the system's state and initializing various hardware components.

3. Diagnostics and Initialization:

- If the diagnostics pass, the bootstrap program proceeds with initializing critical system components, including CPU registers, device controllers, and main memory (RAM).

- It ensures that the system is in a stable state for further operations.

4. Locating the Operating System:

- The bootstrap program's next task is to locate the operating system's kernel.

- In some systems, the entire operating system is stored in ROM (firmware), making it easy to access.

- Other systems may load a small, sophisticated boot program from disk (e.g., from block zero) to initiate the booting process.

5. Loading the Operating System:

- Once the bootstrap program identifies the kernel's location, it loads the operating system's kernel into the main memory (RAM).

- This step may involve copying the kernel from firmware or reading it from a boot block on the disk.

- The bootstrap program may also perform additional checks or operations to prepare for kernel execution.

6. Starting the Operating System:

- With the operating system's kernel in RAM, the bootstrap program hands over control to the kernel.

- The kernel initializes itself, sets up data structures, and begins managing the computer's resources.

- The system is now considered to be running, and the kernel is responsible for executing user programs and managing hardware resources.

It's important to note that the specific details of the boot process can vary depending on the computer architecture and the operating system in use. Additionally, modern systems may involve more complex bootloaders and multiple stages of loading, especially in the case of PCs with various hardware configurations. For example, GRUB is a popular bootloader used in many Linux systems, providing flexibility and configurability during the boot process.