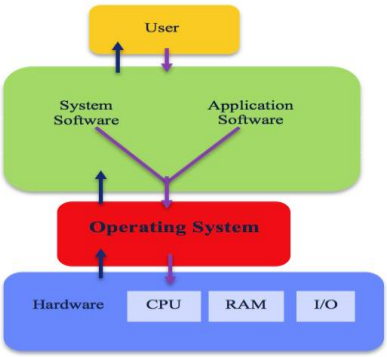
FUNCTIONS OF OPERATING SYSTEM

An operating system is a program that acts as an interface between the computer user and computer hardware, and controls the execution of programs.

The operating system (OS) manages all of the software and hardware on the computer. It performs basic tasks such as file, memory and process management, handling input and output, and controlling peripheral devices such as disk drives and printers.

Most of the time, there are several different computer programs running at the same time, and they all need to access your computer's central processing unit (CPU), memory and storage. The operating system coordinates all of this to make sure each program gets what it needs.



Operating systems performs the following functions:

1. **Process Management**

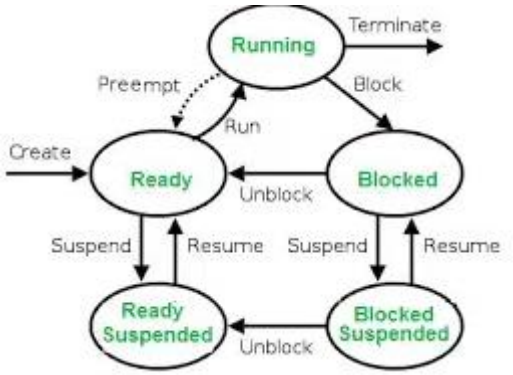
Process management involves creating, scheduling, and terminating processes. The kernel handles these tasks to ensure that multiple processes can run concurrently without interfering with each other.

Process management can be thought of as an instance of a program that is currently executing on a computer system. Managing processes is an important aspect of any

modern operating system, as it allows multiple programs to run concurrently and share system resources efficiently.

The process management OS subsystem is responsible for allocating system resources, such as CPU time, memory, and input/output devices, to running processes. Process management os is also responsible for scheduling the execution of processes in a way that maximizes system throughput and minimizes response times.

Without proper process management OS, a computer system can become unresponsive or even crash if one or more processes consume all available system resources. Therefore, process management OS is a critical component of any modern operating system, and its design and implementation have a significant impact on the overall performance and usability of the system.



Below is a list of some of the activities in process management:

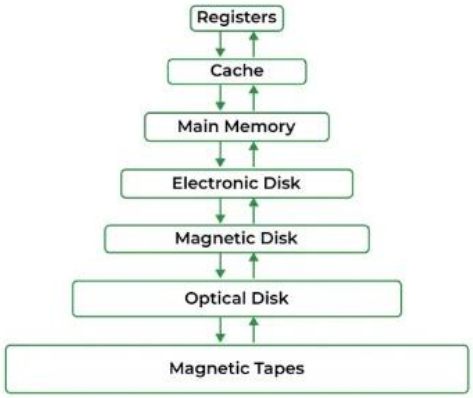
1.1 Process Creation: In Unix-based systems, the fork() system call creates a new process by duplicating the existing one. The exec() family of functions replaces the process's memory space with a new program. In Windows, CreateProcess is used for creating new processes.

1.2 Process Scheduling: The kernel uses scheduling algorithms like Round Robin, Priority Scheduling, or Multi-level Queue Scheduling to allocate CPU time to various processes.

1.3 Process Termination: The exit() system call terminates a process, and kill() can be used to terminate other processes.

2. **Memory Management**

Memory management ensures efficient allocation and deallocation of memory. It includes managing virtual memory, which allows the system to use more memory than physically available by swapping data to disk.



Below is a list of activities in memory management:

2.1 Allocation and Deallocation: In C, malloc() and free() are used for dynamic memory allocation and deallocation, respectively. In kernel space, kmalloc() and kfree() serve similar purposes.

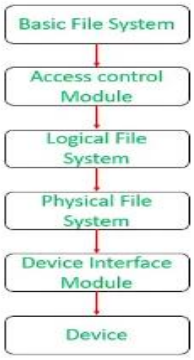
2.2 Paging and Swapping: The system swaps pages in and out of physical memory to disk storage, allowing processes to run as if they have more RAM.

2.3 Memory Mapping: The mmap() function maps files or devices into memory, allowing file I/O to be treated as memory access.

3. **File System Management**

A file system is organized into directories for efficient or easy navigation and usage. These directories may contain other directories and other files. An Operating System

carries out the following file management activities. It keeps track of where information is stored, user access settings, the status of every file, and more. These facilities are collectively known as the file system. An OS keeps track of information regarding the creation, deletion, transfer, copy, and storage of files in an organized way. It also maintains the integrity of the data stored in these files, including the file directory structure, by protecting against unauthorized access.



File system management includes operations for reading, writing, opening, closing files, and managing directories. It ensures data is stored and retrieved efficiently and securely.

3.1 File Operations: open(), read(), write(), and close() are standard Unix system calls for file operations.

3.2 File System Mounting: The mount() system call attaches a file system to a specified directory, while umount() detaches it.

3.3 ***Directory Operations***: mkdir(), rmdir(), and opendir() manage directories.

4. **Device Management**

An OS manages device communication via its respective drivers. It performs the following activities for device management.

- Keeps track of all devices connected to the system. Designates a program responsible for every device known as the Input/Output controller.
- Decide which process gets access to a certain device and for how long.
- Allocates devices effectively and efficiently. Deallocates devices when they are no longer required.
- There are various input and output devices. An OS controls the working of these input-output devices.
- It receives the requests from these devices, performs a specific task, and communicates back to the requesting process.

Device management involves handling I/O operations for hardware devices, managing interrupts, and enabling Direct Memory Access (DMA) for high-speed data transfer.

4.1 Device Drivers: These are specialized programs that control hardware devices. They provide an interface for the OS to interact with hardware.

4.2 Interrupt Handling: The kernel handles interrupts, which are signals from hardware that require immediate attention. An interrupt service routine (ISR) is executed to process the interrupt.

4.3 DMA Management: Allows devices to transfer data to/from memory without CPU intervention, improving efficiency.

5. **Networking**

Networking routines manage network communications, including creating sockets, handling packets, and implementing protocols like TCP/IP.

5.1 Socket Management: System calls like socket(), bind(), listen(), and accept() manage network connections.

5.2 Packet Routing and Forwarding: The kernel routes packets to their destinations based on IP addresses.

5.3 Protocol Handling: The kernel implements protocols such as TCP, UDP, and IP for reliable and efficient data transfer.

6. **Inter-process Communication (IPC)**

IPC mechanisms enable processes to communicate and synchronize with each other. They include pipes, message queues, semaphores, shared memory, and signals.

6.1 Pipes and FIFOs: Pipes allow one-way communication between processes, while FIFOs (named pipes) allow bidirectional communication.

6.2 Message Queues, Semaphores, Shared Memory: These mechanisms provide more complex communication and synchronization options.

6.3 Signals: Lightweight notifications sent to processes (e.g., kill to send signals).

7. **Security**

The operating system uses password protection to protect user data and similar other techniques. It also prevents unauthorized access to programs and user data. The operating system provides various techniques which assure the integrity and confidentiality of user data.

Security routines handle authentication, authorization, and auditing to protect system resources and data.

7.1 Authentication: Validates user identities, typically through login mechanisms.

7.2 Authorization: Enforces permissions and access controls to restrict resource usage.

7.3 Auditing: Logs security-related events and access attempts for monitoring and analysis.

8. **System Calls Interface**

The system call interface provides the means for user applications to request services from the kernel. It includes dispatching system calls and handling errors.

8.1 System Call Dispatching: When an application makes a system call, the kernel dispatches it to the appropriate handler function.

8.2 Error Handling: System calls return error codes to indicate failure, which the application can handle appropriately.